

Developing an Agile-Based Model for Addressing Defects in Medical Device Software Development

Misheck Nyirenda BSc., MSc.

Submitted for the degree of Doctor of Philosophy

School of Informatics and Creative Arts

Dundalk Institute of Technology

Supervised by:

Dr. Martin Mc Hugh

Dr. Róisín Loughran

Professor Fergal McCaffery

Professor Christopher Nugent – Ulster University

PhD

August 2026

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Doctor of Philosophy is entirely my own work, and that I have exercised reasonable care to ensure that the work is original and have conformed to the regulations on the use of and declaration of Generative AI, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: Misheck Nyirenda (Candidate) ID No.: D00252047 Date: 19th August 2026

Acknowledgement

I would like to express my sincere gratitude to my supervisors Dr. Martin Mc Hugh, Dr. Róisín Loughran, Professor Fergal McCaffery, and Professor Christopher Nugent for their unwavering support throughout my studies. I deeply appreciate their encouragement, motivation, inspiration, and invaluable insights, as well as the thoughtful feedback and guidance they offered to me. Certainly, without their timely and unwavering support it could have been difficult for me to conduct and complete my research. Their kindness made me feel very comfortable working with them.

I would like to thank Dr. Fiona Lawless and Dr. Peadar Grant for their unwavering support and trust in allowing me to guide students across various modules throughout the time of my study. This experience has been both fulfilling and invaluable. I also would like to thank Mr. Niall Dowd for his support and trust in involving me in various activities within the Department of Computing Science and Mathematics. Additionally, I am thankful to all my colleagues in the School of Informatics and Creative Arts and the broader DkIT community for their support and encouragement.

I would also like to thank my postgraduate colleagues, both past and present, for their unwavering support and encouragement. I am also deeply grateful to all medical device software organisations who participated in this research. Their dedication and willingness to assist me despite their busy schedules are greatly appreciated. I also would like to thank Breanndán Casey for his invaluable support in reaching out to companies and requesting their participation in the validation process.

I am sincerely grateful to my beloved wife, Mjesa, and my children, Shalom, Ezekiel, and Sibusisiwe for their steadfast love, patience, and endless support and encouragement even during my prolonged absences from their lives. I also thank my family and friends for their constant encouragement and support throughout this journey.

Above all, I thank the Almighty God who makes everything possible for me. His incredible work in the lives of many people for my sake is truly indescribable. Surely, God's grace wins every time.

Table of Contents

Declaration i
Acknowledgement ii
Table of Tables vii
Table of Figures viii
Publications by Candidate ix
Abstract x
Glossary of Terms xi
1 Introduction 1
1.1 Overview 1
1.2 Background and Context 1
1.3 Research Problem 2
1.4 Defect 4
1.5 Defect Prevention and Identification 4
1.6 Research Aim, Questions and Objectives 6
1.6.1 Research Aim 6
1.6.2 Research Questions 6
1.6.3 Research Objectives 7
1.7 Research Approach 8
1.8 Research Contribution 10
1.9 Thesis Structure 10
2 Literature Review 12
2.1 Introduction 12
2.2 Approach Taken and Justification 12
2.3 Medical Device Software 13
2.4 Medical Device Failures and Recalls 14
2.5 Examples of MDS Defects and Their Mapping to IEC62304 Lifecycle Processes 17
2.6 Existing Techniques for Addressing Defects in MDS 24
2.7 Software Development Life Cycle Models 25
2.7.1 Plan-Driven SDLC Models 26
2.7.2 Agile Software Development 30
2.7.3 Agile Scaling Frameworks 60
2.7.4 Lean Software Development 61

2.8 Agile in Medical Device Software Development 63
2.8.1 Agile Practices Adopted in MDS Development 64
2.9 Medical Device Software Regulations and Standards 75
2.9.1 Regulations 75
2.9.2 Standards 77
2.10 Identification of suitable Agile Practices 80
2.11 An Evaluation of Identified Agile Practices in relation to Addressing Defects 84
2.12 Summary 91
3 Research Methodology 94
3.1 Introduction 94
3.2 Research Paradigms 95
3.2.1 Pragmatism 95
3.3 Research Approaches 97
3.3.1 Deductive Research Approach 97
3.3.2 Inductive Research Approach 97
3.4 Research Strategies 98
3.4.1 Design Science Research 98
3.5 Research Choices 101
3.6 Time Horizon 101
3.7 Application of Design Science Research Methodology 102
3.7.1 Activity 1: Problem Identification and Motivation 102
3.7.2 Activity 2: Define the Objectives for a Solution 102
3.7.3 Activity 3: Design and Development 102
3.7.4 Activity 4: Demonstration 103
3.7.5 Activity 5: Evaluation 103
3.7.6 Activity 6: Communication 103
3.8 Research Reliability and Validity 103
3.8.1 Reliability 103
3.8.2 Validity 104
3.9 Survey Procedure 107
3.9.1 Target Population 107
3.9.2 Sampling Technique 108
3.9.3 Sample Size for Survey 108
3.9.4 Ethical Clearance 109

3.9.5 Survey Method 109
3.9.6 Development and Validation of Survey Instrument 110
3.10 Semi-structured Interviews 111
3.10.1 Introduction 111
3.10.2 Developing the Interview Guide and Questions 113
3.10.3 Sample Size for Semi-structured Interviews 114
3.10.4 Selecting Interviewees 115
3.10.5 Profiles of Interviewees 116
3.10.6 Conducting the Semi-structured Interviews 116
3.10.7 Transcribing the Interviews 117
3.11 Summary 117
4 Development of the AV4AD Model 119
4.1 Introduction 119
4.2 Analysis of Agile Practices 119
4.3 Discussion 135
4.4 Agile Practices used for Addressing Defects in GSD and MDS domains 137
4.5 Survey of Medical Device Software Organisations 139
4.5.1 Survey Results Analysis 139
4.5.2 Discussion 142
4.5.3 Survey Limitations 142
4.6 Approach to the Development of the AV4AD Model 143
4.7 Summary 152
5 Evaluation and Validation of the AV4AD Model 153
5.1 Introduction 153
5.2 Evaluation 153
5.2.1 Analysis of the Semi-structured Interviews 153
5.2.2 Interviewees Feedback 154
5.3 Validation 179
5.3.1 Introduction 179
5.3.2 Identifying a Participating Organisation 179
5.3.3 Validation in Organisation A 180
5.3.4 Validation in Organisation B 183
5.3.5 Validation in Organisation C 186
5.3.6 Validation in Organisation D 189

5.3.7 Discussion 192
5.4 Summary 192
6 Summary and Conclusions 194
6.1 Introduction 194
6.2 Summary 194
6.3 Research Questions Revisited 196
6.3.1 Research Objective 1 197
6.3.2 Research Objective 2 197
6.3.3 Research Objective 3 198
6.3.4 Research Objective 4 198
6.3.5 Research Objective 5 199
6.3.6 Research Objective 6 200
6.4 Research Contribution 200
6.4.1 Contribution through Literature Review 200
6.4.2 Contribution through MDS Organisations Survey 201
6.4.3 Model Development 201
6.4.4 Contribution through Conference Papers 202
6.5 Research Limitations 203
6.6 Further Research 204
6.7 Conclusions 205
7 References 206
8 Appendices 257
Appendix A: Research process Steps 257
Appendix B: Studies from the literature review and Analysis 258
Appendix C: Materials for Survey 260
Appendix D: Semi-structured Interview Guide 270
Appendix E: Semi-structured Interview Transcripts 280
Appendix F: Code Book 456

Table of Tables

Table 1.1**:** Research Approach overview 8
Table 2.1**:** Main categories of defects and sub-categories 17
Table 2.2**:** Failure modes and examples 18
Table 2.3**:** SW91 main defect categories and the identified defects 21
Table 2.4**:** IEC 62304 and SW91 (Rajaram *et al.,* 2019) 22
Table 2.5**:** SW91 main defect categories and the identified software defects 22
Table 2.6**:** Primary and Corollary XP practices 39
Table 2.7**:** Agile Modeling Practices 56
Table 2.8**:** Agile Practices adoption in MDS development 72
Table 2.9**:** MDR medical device classification 76
Table 2.10**:** Agile methodologies and agile practices 81
Table 2.11: Agile methodologies, identified practices and excluded similar practices 83
Table 2.12: Agile methodologies and practices not for addressing defects 84
Table 3.1: Summary of research methodology 118
Table 4.1: Definition of terms used in subsequent tables 120
Table 4.2: Summary of PP studies on software defects 121
Table 4.3: Summary of TDD studies on software defects 122
Table 4.4: Summary of Unit Testing studies on software defects 123
Table 4.5: Summary of Refactoring studies on software defects 124
Table 4.6: Summary of Code Review studies on software defects 125
Table 4.7: Summary of CI studies on software defects 126
Table 4.8: Suggested practices that can be used in combination 137
Table 4.9: Articles reporting agile practices for addressing defects in GSD and MDS 138
Table 4.10**:**. Agile practices usage for addressing defects in MDS development 141
Table 4.11**:**. Stages where agile practices are used for addressing defects in MDS development 142

Table of Figures

Figure 1.1: Software-related medical devices recall in the US and the EU (Sedgwick Brand Protection 2026b; Sedgwick Brand Protection 2026a) 3
Figure 1.2: Defect Prevention and Defect Identification 5
Figure 1.3: Overview of Research Approach 9
Figure 2.1: Waterfall Model (Royce 1970) 26
Figure 2.2: The V-Model (Mathur and Malik 2010) 28
Figure 2.3: The Spiral model (Boehm 1988) 30
Figure 2.4: Scrum Model (Lopes de Souza et al. 2021) 36
Figure 2.5: User Story template and example User Story on card (Rubin 2012) 36
Figure 2.6: XP Model (Abrahamsson et al. 2002) 38
Figure 2.7: The XP onion (Wake 2002) 40
Figure 2.8: Family of Crystal Approaches (Abrahamsson et al. 2002) 43
Figure 2.9: FDD process (Abrahamsson et al. 2002) 47
Figure 2.10: DSDM model (Hunt 2006) 50
Figure 2.11: Adaptive Software Development lifecycle (Abrahamsson et al. 2002) 51
Figure 2.12: RUP lifecycle (Kruchten 2000) 54
Figure 2.13: AMDD Lifecycle (Ambler 2007) 57
Figure 2.14: DevOps Lifecycle 59
Figure 2.15: SAFe (Mucambe et al. 2020) 61
Figure 3.1: The Research ‘onion’ by Saunders et al. (2019) 94
Figure 3.2: The DSRM Process Model adopted from Peffers *et al.* (2007) 101
Figure 4.1: The AV4AD Model 145
Figure 5.1: Final AV4AD model 178

Publications by Candidate

Nyirenda, M., Loughran, R., McHugh, M., Nugent, C. and McCaffery, F., 2023, August. Identifying agile practices to reduce defects in medical device software development. In *European Conference on Software Process Improvement* (pp. 61-75). Cham: Springer Nature Switzerland. https://link.springer.com/chapter/10.1007/978-3-031-42310-9_5

Nyirenda, M., Mc Hugh, M., Loughran, R. and McCaffery, F., 2024, July. Using agile practices to address defects in medical device software development: a survey of medical device software organisations in Ireland. In *2024 8th International Conference on Computer, Software and Modeling (ICCSM)* (pp.1- 4). IEEE. https://ieeexplore.ieee.org/abstract/document/10638583

Nyirenda, M., McHugh, M., Loughran, R. and McCaffery, F., 2024, September. An Agile-Based Framework for Addressing Defects in Medical Device Software Development. In *European Conference on Software Process Improvement* (pp. 279-293). Cham: Springer Nature Switzerland. https://link.springer.com/chapter/10.1007/978-3-031-71142-8_20

Abstract

Medical Device Software (MDS) plays an important role in modern healthcare by enabling complex tasks that would otherwise be impossible to perform. However, defects in MDS result in patient injuries, deaths, and recalls of numerous medical devices. These incidents ultimately lead to reputational damage and financial losses for MDS organisations. This research focuses on addressing defects in MDS to avoid these costly incidents.

In software development projects, defects can be addressed through prevention in the early stages of the development lifecycle and identification in later stages. While prevention reduces the risk of defects in later stages and the final release, it is often overlooked or inadequately implemented. MDS organisations emphasise defect identification through software testing when coding is completed. Various software testing techniques such as taxonomy-based testing like IBM's Orthogonal Defect Classification (ODC) are used. However, given the safety-critical nature of MDS, it is essential to address defects throughout the development lifecycle by emphasising on both prevention and identification.

To address this need, this research developed the Agile V-Model for Addressing Defects (AV4AD) model, drawing insights from the literature review and a survey of MDS organisations. The AV4AD model was developed by integrating agile practices that address defects with the V-Model. It is designed for use by any MDS organisation regardless of organisational size and MDS classification or associated risk level. The model was evaluated through semi-structured interviews conducted with eight developers from different MDS organisations in Ireland. Thereafter, the model underwent validation through desk-based implementation in four MDS organisations in Ireland. Findings from both the evaluation and validation indicate that the AV4AD model can assist MDS organisations in addressing defects throughout the MDS development lifecycle, thereby improving overall software quality and reducing the risk of defects slipping to the final release. The findings further indicate the AV4AD model's suitability for adoption in MDS development projects, particularly given its potential to minimise the likelihood of defects that could lead to patient injuries, deaths, financial loss, and reputational damage for MDS organisations.

Glossary of Terms

Agile Software Development (ASD) – An umbrella term for a number of software development methods such as Scrum and XP which adopt iterative development techniques. Agile methods consist of a number of agile practices.

Behaviour-Driven Development (BDD) – A practice that encourages collaboration between developers, analysts, quality specialists, and business stakeholders. It focuses on defining an application's expected behaviour using human-readable, plain language examples, which are then used to automatically test the system's functionality.

Defect – Generically to refer to the inability of a system or component to perform its required functions within specified performance requirements. A defect is also defined as an imperfection or deficiency in a work product where that work product does not meet its requirements or specifications.

Defect Identification – The process of finding defects in software modules with the goal of enhancing the quality of a software system. The systematic practice of finding, logging, and analysing defects after they occur in software.

Defect prevention – The process of improving software quality and productivity by averting the introduction of defects into the software product. The proactive process of detecting and removing the root causes of defects before they occur in a final product.

General Software Domain (GSD) – The broad, foundational realm of mainstream software development that isn't bound by strict safety-critical requirements.

Medical Device Software (MDS) – Software used in the medical domain for a medical purpose(s).

Notified Body – A notified body is a body within the European Union who has been accredited by an EU member state, granting them the ability to award the CE mark, which allows a medical device manufacturer permission to market a medical device for use anywhere within the EU.

Safety Critical Software – Software, whether performing correctly or incorrectly, that have a direct impact on a person’s wellbeing.

Software Defect – An error, flaw, bug, mistake, failure, or fault in a computer program or system that may generate an inaccurate or unexpected outcome or precludes the software from behaving as intended.

Software Development Life Cycle (SDLC) – A model or framework consisting of a number of stages which are required to be completed when developing software. These stages can include Planning, Implementing, Testing and Release.

Test-Driven Development (TDD) – A practice where tests are written to define a feature's requirements before writing the functional code.

User Acceptance Test – A practice where real users evaluate a system in a real-world scenario. Its primary goal is to ensure the software functions as expected and meets all business requirements before its official release.

1 Introduction

1.1 Overview

Defects in Medical Device Software (MDS) can result in patient injuries, deaths, and the recall of numerous medical devices. These issues ultimately lead to reputational damage and financial losses for MDS organisations. This thesis presents the Agile V-Model for Addressing Defects (AV4AD), a model designed to assist MDS organisations to address defects throughout the development lifecycle and mitigate these consequences. This chapter provides an overview of the focus of this research and the context in which it is conducted.

1.2 Background and Context

The work presented in this thesis belongs to the broad field of Software Engineering, specifically within the domain of MDS development. Software has become an integral part of our daily lives and plays a vital role in various aspects, including communication, shopping, finance management, and entertainment (Kosti 2017). It is also used in safety-critical systems where failure could result in severe catastrophes (Addagarrala and Kinnicutt 2018; McGee and McGregor 2016). Examples of safety-critical systems include medical devices, aircraft control systems and nuclear systems. The integration of software in medical devices enhances their capabilities and performance. For instance, software-controlled medical devices can track patients' vital signs (e.g., heart rate, blood pressure etc.), alert physicians to the risk of adverse drug events, and regulate medication dosage for patients (Ronquillo and Zuckerman, 2017; Pashkov and Harkusha, 2020).

Software used in the medical domain for a medical purpose(s) is termed as MDS (Pashkov et al. 2016; IMDRF 2014). MDS must meet regulatory requirements before it can be marketed to ensure its reliability, safety, and user-friendliness (Ivanova and Kadurin 2021; McHugh et al. 2015). In the United States (US), MDS manufacturers must comply with the US Food and Drug Administration (FDA) regulations, while in the European Union (EU), they must comply with the EU Medical Device Regulation 2017/745 (EU 2023; Melvin 2022). It is particularly vital that MDS must be of high quality and reliable to avoid defects that can negatively affect patient's recovery, health, and well-being (Shroff et al. 2011; Fu 2011). Software reliability involves fault avoidance, elimination, and tolerance (Mili et al. 1999), but achieving this can be challenging due to the limitations of software development techniques (Knight and Wika 2010). Moreover, designing and developing high quality, reliable, effective, usable and

innovative MDS is a complex process which comprises many regulatory compliance activities (Hrgarek 2012). Additionally, the increasing complexity of MDS makes designing and development difficult, forcing developers to focus on releasing the software without considering all scenarios (Badanahatti and Pillutla 2020). The rush to be first-to-market with innovative MDS can result in the release of software with defects that emerge during use (Thompson 2018), leading to medical device recalls, injuries, and even death (Leveson 2017). It is vital to prevent and identify defects early in the development process of MDS to avoid such consequences. This research aims to address defects in MDS development. The research seeks to develop a model to assist MDS organisations in addressing defects throughout the development lifecycle.

1.3 Research Problem

MDS has become an essential component of modern healthcare and increasingly enables sophisticated tasks that would otherwise be impossible to perform. Examples include Magnetic Resonance Imaging (MRI) software which distinguishes between normal and diseased tissue to precisely locate cancerous cells within the body, and software for controlling an implantable pacemaker. However, defects in MDS have caused medical devices failures leading to patient injuries and deaths. For example, the Therac-25 radiation therapy machine massively overdosed six people due to software defects, resulting in serious injuries and death (Leveson and Turner 1993). Between July 2022 and July 2024, software-related failures in Mobile Cardiac Outpatient Telemetry devices resulted in 109 patient injuries and 2 deaths (FDA 2025a). MDS defects have also been a major cause of medical devices recalls in both the US and the EU. In 2011, MDS defects resulted in 778 medical devices recalls with 14 being of high risk (Alemzadeh et al. 2013a). In 2024, 111 medical devices were recalled in the US and 504 in the EU due to software defects (Sedgwick Brand Protection 2024). The number of software-related recalls increased in 2025, rising to 137 recalls in the US and 586 recalls in the EU (Sedgwick Brand Protection 2026b; Sedgwick Brand Protection 2026a). This indicates a continued upward trend in software-related medical device recalls especially in the EU. Figure 1.1 presents an overview of medical device recalls in the US and the EU over the last six years due to software defects, based on data from Sedgwick's quarterly reports such as the 2025 Quarter 4 US recall report (Sedgwick Brand Protection 2026b; Sedgwick Brand Protection 2026a).

Addressing defects throughout the MDS development lifecycle is crucial to avert catastrophic incidents and ensure reliability and safety. Defects that escape early stages can be more costly to find and remove later (Williams and Kessler 2001; Ramdoo and Huzooree 2015). Moreover, post-release defects in MDS may cause injury or death of patients, ultimately damaging the manufacturer's reputation and finances. Boehm and Basili (2001) recommend resolving defects early in the requirements and design phases, as it can be 100 times more expensive after delivery. This suggests that investing into early defect detection and removal could save manufacturers from higher rework costs (Winkler and Biffl 2006).

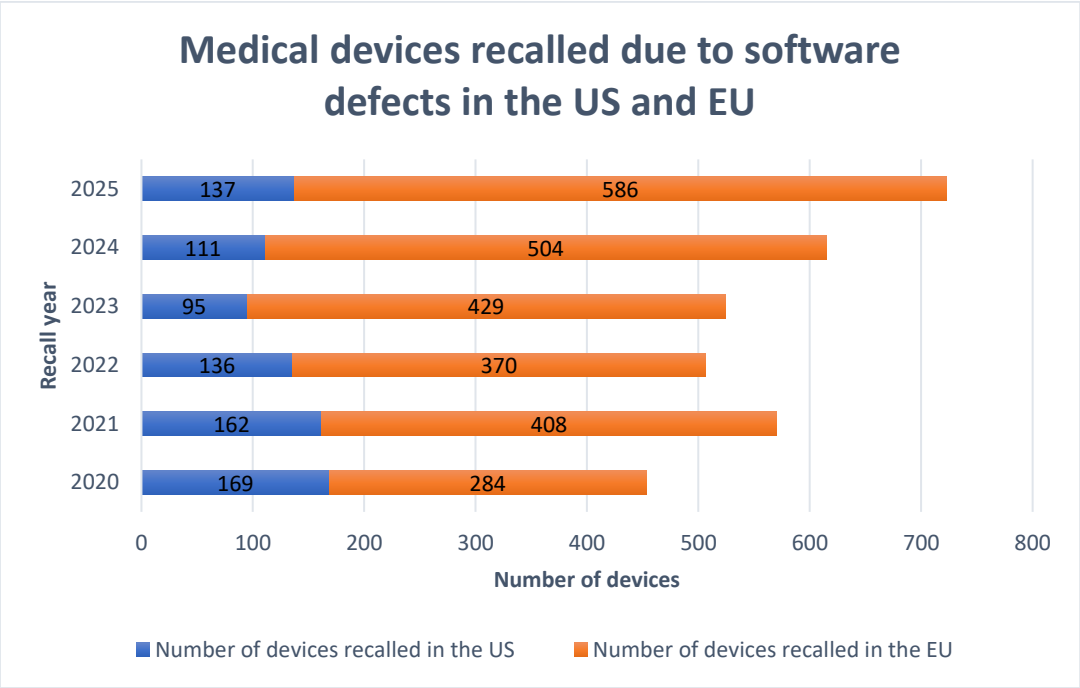


Figure 1.1: Software-related medical devices recall in the US and the EU (Sedgwick Brand Protection 2026b; Sedgwick Brand Protection 2026a)

Agile Software Development (ASD) can prevent expensive reworks and defects in released software (Fogarty et al. 2020; Duka 2012). However, the literature reveals that regulatory constraints limit the exclusive use of ASD methodologies in MDS development (McHugh, McCaffery and Casey 2012; Myklebust et al. 2016). Consequently, agile practices have been integrated into plan driven approaches in MDS development (Rasmussen et al. 2009; Heeager and Nielsen 2020; McHugh et al. 2013). Despite this integration, research about the use of agile practices to address defects in MDS development is scant. It is essential to identify agile practices that can address defects and help developers utilise them to address defects in MDS development. This research investigates agile practices that can address defects in MDS development and develops a model that integrates these agile practices to support defect

prevention and defect identification throughout the development lifecycle. The objective is to improve software quality by reducing the risk of defects slipping to the final release, where they could result in patient harm. The proposed model is designed to assist any MDS organisation seeking to address defects, regardless of organisational size, MDS classification, or associated risk level.

1.4 Defect

The literature reveals that a defect can be referred to by different terms such as bug, fault, error, flaw, failure, and anomaly. These terms are defined differently by various authors. For example, Pandey (2019) defines a defect as "an indication of the failure of meeting requirements or specification due to a deficiency in the work product". Freimut (2001) states that "the term defect is used in a generic manner referring to either a fault or a failure, provided the distinction is non-critical". On the other hand, "failure indicates the inability of the software product to execute the specified function" (Pandey 2019). The same author also mentions that "software failure is the result of a flaw in the design". Some studies show that the term defect can be used generically to refer to a fault or failure (Braga et al. 2015). Research by Dudila and Letia (2013) suggests that the terms defect, bug, fault, error, failure, flaw and anomaly are used generically to refer to the "inability of a system or component to perform its required functions within specified performance requirements". According to Rawat and Dubey (2012) "a software defect is an error, flaw, bug, mistake, failure, or fault in a computer program or system that may generate an inaccurate or unexpected outcome or precludes the software from behaving as intended". Based on these definitions, the terms defect, error, bug, flaw, mistake, failure, fault, and anomaly were used interchangeably to gather relevant literature reviewed and analysed in this research. These terms are also used interchangeably in this research.

1.5 Defect Prevention and Identification

In this research, the term "address defects" or "addressing defects" is used to refer to either defect prevention, defect identification or both. Defect prevention is the process of improving software quality and productivity by averting the introduction of defects into the software product (Memon et al. 2018; Huang and Liu 2017). It is a proactive strategy that focuses on stopping bugs before they are introduced into the software. This approach usually happens in the early stages of the software development lifecycle, including requirements gathering and analysis, system design, architectural design, module design, and implementation (Chakravarty

and Singh 2022; Ergasheva and Kruglov 2020). Addressing defects during these earlier stages reduces correction costs compared to finding them in later stages of the development lifecycle. Additionally, defect prevention reduces the risk of defects persisting into the final software release, thereby lowering the likelihood of catastrophic events. While defect prevention is crucial in software development, it is often overlooked or improperly executed using ineffective methods (Jamal et al. 2015).

Defect identification is the process of detecting defects in software modules with the goal of enhancing the quality of a software system (Mahmoud and Santos 2021). It is a reactive activity that focuses on finding bugs after they have been introduced into the software. This process occurs in the later stages of the software development lifecycle, including implementation, testing, release, and maintenance (Chakravarty and Singh 2022; Ergasheva and Kruglov 2020). An exclusive reliance on defect identification can lead to high rework costs and increase the risk of defects persisting into the final software release. These processes are illustrated in figure 1.2 in the form of the V-Model which is regarded as the popular choice for developing safety-critical software like MDS, because of its capability to produce deliverables that are essential for regulatory approval (Hauschild et al. 2022; Liu et al. 2016; McHugh et al. 2014).

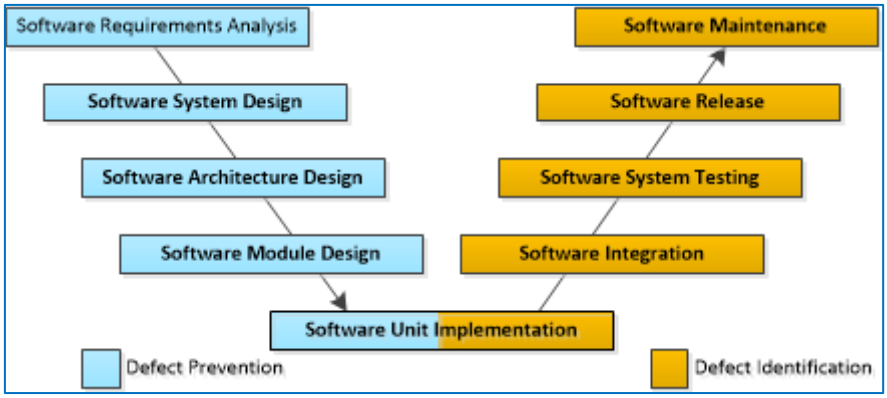


Figure 1.2: Defect Prevention and Defect Identification

It is crucial to ensure safety throughout the development lifecycle of MDS. Implementing a proactive approach of defect prevention in the early stages of MDS development lifecycle can reduce the risk of defects in MDS. This ensures that defects are caught early on, preventing their occurrence in later stages where they can lead to expensive rework (Ahmed Khalid and Yeoh 2021; Kotagi and Yadav 2023). Moreover, defect prevention in early stages lowers the risk of defects slipping into the final release of the software where they are costly to find and fix (Li et al. 2023).

Despite the widely advocated benefit of producing high-quality software through defect prevention in the early stages of software development lifecycle (Chakravarty and Singh 2022), many software development projects often neglect or inadequately implement this important step (Jamal et al. 2015). Considering the safety-critical nature and the increasing complexity of MDS to support a growing number of functionalities, prioritising defect identification while neglecting defect prevention, increases the risk of defects slipping through to the final release of the software. This arises due to the increased likelihood of developers missing the critical variables associated with the medical environment (Critical Software 2020). The growing complexity of MDS poses significant challenges in ensuring its safety. Therefore, it is essential in MDS development to address defects throughout the development lifecycle. Prioritising and practicing defect prevention during the early development stages is particularly beneficial, as defects become increasingly costly to find and fix the longer they remain in the development lifecycle (Felix and Lee 2020). This delay increases the likelihood of defects slipping to the final release. These defects become critical in medical devices whose operational functionality relies on the effectiveness and reliability of the software, potentially leading to patient harm.

1.6 Research Aim, Questions and Objectives

1.6.1 Research Aim

This research aims to address defects in MDS by using agile practices to prevent them in the early stages of the development lifecycle and identify those that may occur in later stages. This approach averts costly rework and potential catastrophes that may arise from defects occurring during use. Therefore, the primary focus of this research is the development of the AV4AD, a model that integrates agile practices for addressing defects with the V-Model, to assist MDS organisations to address defects throughout the development lifecycle.

1.6.2 Research Questions

To achieve the stated aim, this research will answer the following three Research Questions (RQ):

RQ1: *Which agile practices can address software defects?*

RQ2: *Which agile practices are used to address defects in MDS development?*

RQ3: *How can agile practices be integrated in MDS development to address defects?*

1.6.3 Research Objectives

In pursuit of the stated aim and to address the research questions, the research seeks to achieve the following Research Objectives (RO):

RO1: To identify agile practices that have been used to address software defects in the General Software Domain (GSD).

RO2: To identify which agile practices have been used in MDS development. This objective focuses on the general use of agile practices in the MDS domain.

RO3: To identify agile practices that have been used for addressing defects in MDS development. This objective focuses on whether any agile practice(s) was specifically used for addressing defects in MDS development.

RO4: To investigate which agile practices used for addressing defects in GSD can also be used to address defects in MDS development.

RO5: To develop and evaluate a model that incorporates agile practices to address defects throughout the MDS development lifecycle.

RO6: To validate the model.

An overview of how the Research Questions and Research Objectives have been addressed in the course of this research is provided in table 1.1.

Table 1.1**:** Research Approach overview

Research Question	**Research Objective**	**Research Method**
RQ1: Which agile practices can address software defects?	**RO1:** To identify agile practices that have been used to address software defects in GSD.	Literature review
RQ2: Which agile practices are used to address defects in MDS development?	**RO2:** To identify which agile practices have been used in MDS development.	Literature review & Survey
	RO3: To identify agile practices that have been used for addressing defects in MDS development.	
	RO4: To investigate which agile practices used for addressing defects in GSD can also be used to address defects in MDS.	Literature Review & Analysis
RQ3: How can agile practices be integrated into MDS development to address defects?	**RO5:** To develop and evaluate a model that incorporates agile practices to address defects throughout the MDS development lifecycle.	Design & Development, Semi-structured Interviews
	RO6: To validate the model.	Industry implementation

1.7 Research Approach

This section outlines the research approach used to address the research questions presented in Section 1.6.2 and achieve the objectives defined in Section 1.6.3. To answer RQ1, a literature review was conducted in four stages:

- The first stage focused on investigating the problem of defects in MDS and its development processes, as well as regulatory standards.
- The second stage focused on gathering agile practices from commonly used agile methodologies and examining them for their potential to address defects.
- The third stage investigated the use of these agile practices for addressing defects in GSD.
- The fourth stage investigated the general adoption of agile practices in MDS development and whether any were specifically used for addressing defects. A further

search was conducted to find studies that specifically reported using agile practices to address defects in MDS development. This was necessary to gather important insights into how the practices were implemented, as the majority of studies reviewed in the first part of the fourth stage did not cover the aspect of using a particular agile practice to address defects. Finally, an analysis of each of the identified agile practice in stages two and three was performed to determine their potential to address defects in MDS development.

RQ2 was addressed through a literature review that focused on investigating the use of agile practices specifically for addressing defects in MDS development. This literature review was complimented by a survey of MDS organisations in Ireland aimed at gathering insights into which agile practices are currently used for addressing defects in MDS development and how they are implemented.

RQ3 was addressed through development, evaluation and validation of the AV4AD model. The development of the model was based on the insights gained through the literature review and the survey. The AV4AD model underwent evaluation through semi-structured interviews with MDS developers. Thereafter, the AV4AD model was validated through implementation within MDS organisations. A graphical overview of the Research Approach to address Research Questions and Objectives developed in this study is shown in figure 1.3. An overview of the research process steps can be found in Appendix A.

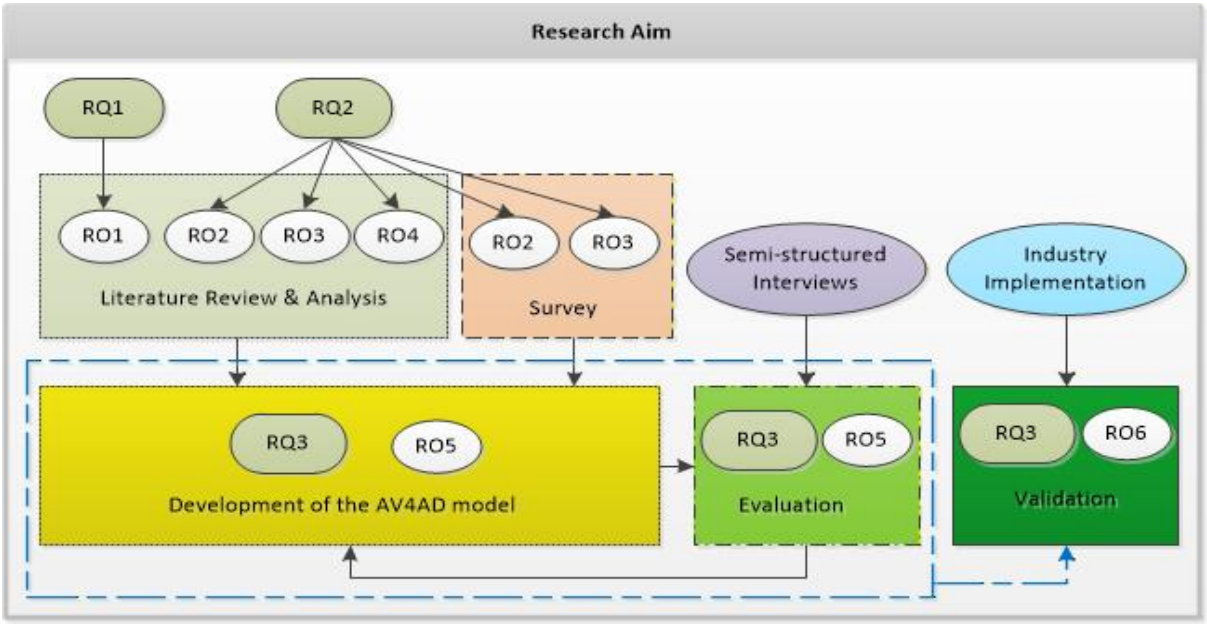


Figure 1.3: Overview of Research Approach

1.8 Research Contribution

As MDS increasingly plays a key role in healthcare, ensuring its safety and reliability is paramount to prevent patient injuries and deaths. It is crucial to prevent defects early in the development life cycle of MDS to avoid costly reworks and catastrophic events. This research aims to assist MDS manufacturers in using agile practices to prevent defects in the early stages of development lifecycle and identify those that may arise later. The research makes the following contributions:

1. The primary contribution of this research is the AV4AD model. The AV4AD is designed to assist MDS organisations in addressing defects by preventing them in the early stages of MDS development lifecycle and identifying those that may arise in later stages. The model provides an iterative process at each stage and includes the flexibility to correct a defect from any previous stage. For instance, if a defect is detected during the software detailed design stage and requires correction at the software architectural design stage, it will be corrected at the software architectural design stage before proceeding to the software unit implementation stage. Similarly, a defect detected during software unit implementation requiring correction at the software requirements analysis stage, will be corrected at the software requirements analysis stage before proceeding to the software unit testing stage. To provide evidence of the AV4AD model's suitability and effectiveness in addressing defects, it was evaluated through semi-structured interviews and validated within two MDS organisations. The AV4AD model has been published in international conference proceedings.

2. An extensive literature review that identified and analysed a list of agile practices to determine their applicability to address defects in MDS development. Consequently, this research provides MDS organisations with a list of agile practices to address defects during the development of MDS.

3. The publication of knowledge gathered during the research (3 publications).

1.9 Thesis Structure

This section outlines the structure of the remaining chapters of this thesis. Chapter 2 presents the literature review conducted in the initial stages of this research. It provides insights into the development of MDS and its role on modern healthcare. The chapter also reviews software

development lifecycle models, MDS regulations and standards. Chapter 3 outlines the research methodology used in this study to achieve the research objectives. It provides an overview of various components that form the research methodology and justification for the choices made. The chapter also presents the survey and semi-structured interview processes. Chapter 4 focuses on the development of the AV4AD model. The chapter begins with the analysis of agile practices to determine their applicability to address defects in MDS development. It then presents the survey results, followed by a discussion. Finally, the chapter details the integration of agile practices for addressing defects with the V-Model, forming the AV4AD model. Chapter 5 presents the evaluation and validation of the AV4AD model. Evaluation was conducted through semi-structured interviews with MDS developers from various organisations in Ireland. Validation was done through implementation in two MDS organisations in Ireland. Chapter 6 concludes the study, summarising the main aspects of this research and reflecting on the findings, contributions, limitations, and directions for future research.

2 Literature Review

2.1 Introduction

This chapter presents the literature review conducted in this research, forming the foundation for the remainder of this study. The initial stage involved an exploratory study focused on finding existing problems within the research context. The literature review provides insights into the development of MDS and examines the problem of defects associated with its implementation. The chapter begins with the approach taken whilst conducting the literature review. It then provides an overview of MDS, outlining its increasing role in healthcare. Thereafter, the chapter outlines the problems resulting from defects in MDS and outlines examples of defects that occur in MDS. The chapter then reviews existing techniques for addressing defects in MDS development. Various software development lifecycle models that could be used for MDS are examined. This is followed by a review of agile development in the MDS development domain, highlighting the adoption of agile practices. The chapter also presents an overview of regulations and standards relevant to MDS development. Thereafter, an initial literature review is performed to identify potential agile practices from various agile methodologies, which are then individually assessed to ascertain their potential to address defects. The chapter concludes with a summary of key insights from the review.

2.2 Approach Taken and Justification

The literature review in this study takes a non-systematic approach, which is well suited to exploratory and design-oriented research (Webster and Watson 2002). Non-systematic reviews "offer flexibility to address more far-reaching questions and pursue novel insights" (Cook 2019). This significantly enables the researcher to build a broad and contextually grounded understanding of the research topic (Chukwuere 2023). Furthermore, the flexibility offered by non-systematic literature review is particularly important for establishing a comprehensive background for understanding existing knowledge and highlighting the significance of new research (Lau and Kuziemsky 2016). By not adhering to the rigid constraints of systematic review protocols, a non-systematic literature review approach allows the researcher to explore diverse sources, including grey literature, standards documents, industry reports, or methodological papers that may not be captured through rigid systematic protocols (Snyder 2019; Grant and Booth 2009). This is particularly important in the MDS domain, where the urgency of addressing software defects means that critical insights often emerge from practice-

oriented or non-traditional sources. Consequently, non-systematic reviews provide the flexibility needed to identify the key concepts and influential findings that have shaped the debate in the domain to date (McDougall 2015).

Moreover, non-systematic literature reviews allow the researcher to incorporate seminal works and influential studies that may fall outside the scope of systematic search strategies but are essential for understanding the intellectual foundations of a field (Baumeister and Leary 1997; Snyder 2019). Such breadth is vital for exploratory research designs, including design science, where the aim is to understand the landscape, identify gaps, and justify the relevance of the problem under investigation (Webster and Watson 2002). Given that the initial stage of this research involved an exploratory study focused on finding existing problems within the research context, a non-systematic literature review was chosen as the most appropriate approach for the literature review in this study. It provided the necessary flexibility, depth, and contextual insight to inform the development of the research problem and guide subsequent stages of the study.

2.3 Medical Device Software

This section provides a perspective on the use of MDS in the healthcare, underscoring the motivation of this research. Additionally, the section highlights the important role of MDS in enhancing patient care and the risks associated with defects in MDS. This section also discusses two types of MDS and the regulatory frameworks governing its development and marketing.

The rise in disease prevalence and shortage of caregivers coupled with the need to improve healthcare delivery has led to the development and use of innovative technological solutions in healthcare to save human life (Lee et al. 2006; Alotaibi and Federico 2017). Despite advances in medical technology, the lack of proper integration and operation of these systems was identified as a major factor that resulted in systematic inefficiencies in healthcare delivery (Lee et al. 2006). Consequently, we have witnessed the rise in development and use of MDS that enabled the integration and interoperability of different healthcare systems. Most importantly, many medical devices are now software driven which enable them to perform sophisticated functions. Software used in healthcare can be recognised as a medical device on its own (McHugh et al. 2011; IMDRF 2014; Software as a Medical Device Working Group 2017; Pashkov and Harkusha 2020).

There are two types of MDS used in healthcare: Software as a Medical Device (SaMD) and Software in a Medical Device (SiMD) (Gordon and Stern 2019). SaMD is standalone software intended for medical purposes without being part of a hardware medical device (IMDRF 2014; Pashkov et al. 2016). The FDA specifies that "software running on a hardware medical device qualifies as SaMD only if it does not drive or control the medical device" (FDA 2020). On the other hand, SiMD refers to software embedded in traditional medical devices that supports their functionality (Gordon and Stern 2019). Unlike SaMD, SiMD controls the hardware and can present direct harm to patients and users if the software fails (MDCG 2019; Kitawaki et al. 2016). An example of SaMD is a mobile application that tracks a patient's heart rate and provides treatment advice to either the patient, the doctor, or both. An instance of SiMD is software that enables patients to control their insulin pump and regulate their glucose levels. The intrinsic usage of software in healthcare means that a defect can result in a catastrophe. Due to the potential for catastrophic consequences, exceptional care is required in the development of MDS to ensure reliability and safety. Reliable and safe MDS should not contain defects. This research aims to address defects in MDS by using suitable agile practices to prevent them early on and identify those that may occur later in the development process.

2.4 Medical Device Failures and Recalls

As mentioned in section 1.3, defects in MDS can lead to severe consequences, including serious injury or death. For instance, between 1985 and 1987, a software-controlled Therac-25 radiation therapy machine overdosed six patients resulting in severe injuries and death (Leveson and Turner 1993). From 2017 to 2021, 51 serious injuries and 3 deaths resulted from software defects in infusion pumps (FDA 2022a). Between July 2022 and July 2024, software-related failures in Mobile Cardiac Outpatient Telemetry devices resulted in 109 patient injuries and 2 deaths (FDA 2025a). MDS defects have also remained the top cause of medical devices recalls. "Recalls are an indicator of major device problems that have the potential to negatively affect patient safety and/or device effectiveness" (Makower et al. 2010). According to the FDA, a product recall is a "Method of removing or correcting products that are in violation of laws administered by the FDA" (FDA 2025b). From 1983 to 1997, 383 medical devices were recalled because of software defects (Wallace and Kuhn 2001). Radiology devices had the highest recall rate (30%), attributed to the complexity involved in developing systems used in radiology. From 1999 to 2005, there were 1261 software-related medical device recalls, with 425(33.7%) due to software failures (Bliznakov et al. 2007). The recall rate significantly increased compared to the 6% reported by Wallace and Kuhn (2001).

FDA software-related recalls increased from 25% in 1999 to 2000 to 30% in 2001 to 2003, and to 50% in 2004 to 2005 (Bliznakov et al. 2007). Analysis of FDA recall data from 1999 to 2010 found that "four out of every ten medical devices comprising software failed due to software defects" (Bliznakov et al. 2014). Connor et al. (2017) found that 49% of the 502 radiation oncology devices recalled by the FDA from 2002 to 2015 were due to software defects. Simone (2013) found that 1,122 (19.4%) of 5,792 medical devices recalled by the FDA from 2005 to 2011 were software-related. Alemzadeh et al. (2013) reported that from 2006 to 2011, a total of 778 medical devices were recalled due to software issues. From 2009 to 2014, there were 712 software-related recalls out of 6,393 medical devices recalled (Kitawaki et al. 2016). Ronquillo and Zuckerman (2017) found that 12 out 627 software-related medical devices recalled from 2011 to 2015 were high-risk. Eleven of these entered the market through the FDA review process without requiring evidence of safety or effectiveness, and one was fully exempted from regulatory review. Zhang et al. (2019) found that user interface software errors accounted for one-half of all software errors in 423 medical devices recalls from 2012 to 2015.

Z. Fu et al. (2017) found that 82% of 100 software-related medical devices recalled by the FDA from 2014 to 2016 posed a moderate risk to health. From 2017 to 2019, 10.13% of 4,216 medical ultrasound device failures were due to software defects (Wang et al. 2022). Analysis of medical devices recalls by the Therapeutic Goods Administration (TGA) of Australia found that in the five years to April 2020, software defects accounted for over 20% of all recalls (Therapeutic Goods Administration (TGA) 2020). For example, in the period from 1 October 2019 to 1 April 2020, 50 medical devices were recalled due to software defects. In 2021, in the US, software issues remained the top cause of medical devices failures accounting for 162 (19.4%) medical device recalls (Sedgwick Brand Protection 2022b). In the EU, software was the top cause of medical devices recalls in quarter three and quarter four of 2021 and was the second top reason for all medical devices recalls in 2021, accounting for 408 (19.8%) medical devices recalls (Sedgwick Brand Protection 2022a). In the US, from Jan 2018 to June 2022, 10.6% of 189 Class I medical device recalls were due to software defects (Mooghali et al. 2023). From 2015 to 2021, 2,605 ventilators and 277,450 infusion pumps were recalled in the US due to software defects, causing serious injuries and deaths (FDA 2022a; FDA 2022b). From January 2023 to July 2023, 56 medical devices were recalled due to software defects (FDA 2023a). Twelve of the 56 were Class I recall (could cause serious injury or death) and the rest were Class II recall (might cause serious injury or temporary illness). In 2024, there were 111 software-related medical devices recalls in the US and 504 in the EU (Sedgwick

Brand Protection 2024). The number of software-related recalls increased in 2025, rising to 137 recalls in the US and 586 recalls in the EU (Sedgwick Brand Protection 2026b; Sedgwick Brand Protection 2026a). This indicates a continued upward trend in software-related medical device recalls especially in the EU. Medical device-related injuries, deaths, and recalls due to defects in MDS can lead to financial and reputational damages for MDS manufacturers.

As shown in Chapter 1, Figure 1.1, the trend in medical device recalls due to software defects over the last six years reveals contrasting patterns between the US and the EU. In the US, recalls generally decreased from 2020 to 2023 before increasing in 2024 and 2025. On the other hand, recalls in the EU have generally increased over the same period. The existing literature does not provide definitive explanations for these differences. Nonetheless, Schneider and Müller-Sägebrecht (2024) suggest that lessons drawn from analyses of past recall data helps organisations to refine their risk and quality management processes, thereby minimising the likelihood of future recalls. This observation appears particularly relevant to the US where recall data has been publicly accessible for many years. In contrast, public availability of recall data in the EU has only improved recently. Access to past recalls and adverse events reports can help organisations to identify risks, learn from failures, and implement appropriate improvements that potentially lead to improved product quality. On the other hand, Craig et al. (2019), Yoon et al. (2019), Yadav et al. (2025), and Everhart et al. (2025) highlight issues of underreporting or non-reporting which obscure the true extent of device problems and may delay or prevent necessary recalls.

Regarding the increasing number of recalls in the EU, Nüssler (2023) suggest that this trend may be attributed to the stringent regulatory scrutiny under the Regulation (EU) 2017/745 (EU 2023) and notified bodies which are responsible for assessing product conformity with applicable regulations and standards. While recalls may negatively affect manufacturers reputation, they are a source of learning for future quality improvements. Moreover, recalls may symbolise the manufacturers' commitment to safeguarding patient health and adherence to regulations. It is important to proactively address defects in MDS development to avert adverse consequences for patients and financial as well as reputational losses for manufacturers.

2.5 Examples of MDS Defects and Their Mapping to IEC62304 Lifecycle Processes

This section provides examples of software defects that occurred in medical devices and maps these defects to the corresponding lifecycle processes of the IEC62304 standard. A critical review of the literature, including studies analysing medical devices recall data from sources such as the FDA recall database, was performed. The review revealed several categories of software defects that occur in medical devices. For example, Fu *et al.* (2017) classified four major categories of software failures most frequently observed in medical devices: Control Flow Fault, Calculation Fault, System Integration Fault, and Human-Machine Interaction Fault. These authors further subdivided each of the four categories as summarised in table 2.1.

Table 2.1**:** Main categories of defects and sub-categories

Main category	**Sub-categories**
Control Flow Fault	Inconsistent logics from requirements
	Exception handling fault
	Block or unblock interrupts fault
	Execution orders of functions fault
	Conditional statement fault
Calculation Fault	Incorrect arithmetic/formula
	Incorrect/outdated constants
	Incorrect conversion
	Incorrect approximation/precision
System Integration Fault	Mismatch of reused component
	Mismatch of component interfaces
	Inconsistent system evolution
	Mismatch of components configurations
Human-Machine Interaction Fault	Missing/wrong functions of human-machine interactions
	Missing/misleading/confusing/error information
	Inappropriate use of keyboard/button

Alemzadeh *et al.* (2013) categorised the failure modes associated with medical device recalls due to software defects and provided examples for each category summarised in Table 2.2.

Table 2.2**:** Failure modes and examples

Failure mode	**Examples failures**
Alarm or Message Error	Alarm reset Lack of audible alarms Missed alarms Unexpected/false alarms
Physical Safety Hazard	Electrical shock Smoke, fire, or explosion Unintended movement Overdose or overexposure
Display or Image Error	Blank image Display freeze Image distortion/corruption Loss of image data
Treatment Interruption or Therapy Failure	Delayed/failed shock delivery Infusion or ventilation failure Signal analysis failure Loss of monitoring
Device Operation Failure	Device inoperable Failure at startup Failure to stop exposure Hang-up or freeze
Calculation or Output Error	Corrupted patient files Inconsistent output Incorrect calculation or display Miscalculation

Wallace and Kuhn (2001) analysed medical device recall data from the FDA database for the period 1983 to 1997 and identified the following categories of software defects:

1) calculation faults - (incorrect change to counting, overflow, Constants, or table of constants incorrectly coded, Improper handling of data structure).
2) change impact faults - (Logic (incorrect conditions or incompatibility with sequential relationship).
3) configuration management faults - (Software incompatible with other components).
4) data faults - (units, assigned values or problems with the actual input data, System failed due to invalid input data).

5) fault tolerance - (Excessive use of the program causes failure, Incorrect action due to operator error).
6) initialization faults - (Lack of initialization of the runtime environment while the program initially executes or restarts).
7) interface faults - (Software does not properly interface with external device or other software component).
8) logic faults - (incorrect logic in the requirement specification, unexpected behaviour of two or more conditions occurring simultaneously, improper limits).
9) omission faults - (Vital system functions are missing, Lack of documentation, or improper documentation).
10) quality assurance faults - (the processes were not sufficient, or a new version was not validated).
11) requirements fault - (Exceptional conditions were not specified in the requirement specification, Functions missing in the requirement specification, Requirement specification was incorrect for its usage with other components, Test hooks or monitors were not specified).
12) timing faults - (Two inter-react processes are out of time synch with one another, Real time clock was not accurate, scheduled event did not occur due to timer failure).
13) other (includes problems in COTS, EPROM, hardware, resources (e.g., memory), configuration management, typos, mistakes in translating requirements into code, and quality assurance).

Simone (2013) presents several examples of software defects and the corresponding medical devices that were recalled due to these defect, as summarised below:

1) Linear accelerator imaging software was recalled due to inadequate software requirements. The determined root cause in this case was the incomplete software requirements documentation which failed to address possible exception pathways.
2) Radiotherapy treatment planning software used with Linear accelerators was recalled due to inadequate validation of the off-the-shelf software.
3) Vision-correction software for an excimer laser system was recalled due to inadequate device software coding and inadequate installation software for use in upgrades.

4) A cardiac and vascular fluoroscopic X-ray imaging system was recalled for both a hardware defect and an inadequate software mitigation.

5) A bedside monitor was recalled because of these devices at their maximum volume setting may result in the premature failure of the internal speaker.

Geraldi and OliveiraJr (2017) argue that defining defect types is essential for identifying persistent faults in different software artefacts. Moreover, pre-defined defect types are useful for guiding the detection of potential defects. They identify six main types of defects commonly applied to functional requirements and use cases diagrams:

1) Omissions - absence of a mandatory element or functionality, e.g., variations of a certain requirement not presented in the specification
2) Incorrect Facts - a functional requirement or use cases incorrectly described
3) Inconsistencies - problems from functional requirements or use cases with their goals and specifications poorly designed - e.g., descriptions, variations, and terminology
4) Ambiguities - a specified functional requirement or use case does not meet its objective - e.g., descriptions with multiple interpretations or misdescribed
5) Extraneous Information - functional requirements or use cases are redundant - e.g., are duplicated and without specification
6) Consequences or Others - unexpected problems in the specification of functional requirements or use cases - e.g., communication between analysts and developers is flawed with respect to the project objectives and what should or not be done.

Using the key words associated with the identified software defects, a search was performed in the SW91 – Classification of Defects in Health Software standard to determine their corresponding classification. For each defect that aligned with an SW91 categories, the main category defined in the standard was adopted. For example, the software defect ‘missing requirements’ corresponded to the Requirements (2) main category within the SW91. Table 2.3 shows the identified defects mapped to their respective SW91 main categories. As can be seen from table 2.3, most of the identified software defects are linked to SW91 main categories of Requirements (2) and Architectural and Design (3). There were no defects linked to the SW91 main category of Planning (1).

Table 2.3**:** SW91 main defect categories and the identified defects

SW91 Defect category	**Identified software defects**
Planning (1)	
Requirements (2)	Inconsistent logics from requirements, Incorrect logic in requirements specification, Improper limits, Unexpected behaviour of two or more conditions occurring simultaneously, Vital system functions are missing, Lack of documentation or improper documentation, Missing/wrong functions of human-machine interactions, Alarm or message error, Display or image error, Quality assurance faults e.g. the processes were not sufficient, Exceptional conditions were not specified in requirements specification, Function missing in the requirements specification, requirements specification was incorrect for its usage with other components, Test hooks or monitors were not specified, omissions, incorrect facts, inconsistencies, ambiguities, erroneous information, consequences
Architecture and Design (3)	Block or unblock interrupts fault, Incorrect approximation/precision, Mismatch of reused components, Mismatch of component interfaces, Inappropriate use of keyboard/button, Timing faults, Fault tolerance e.g., incorrect action due to operator error, Interface faults e.g., software does not properly interface with external device or other software components
Implementation (4)	Exception handling fault, Execution orders of functions fault, Conditional statement fault, Incorrect arithmetic/formula, Incorrect /outdated constants, Incorrect conversion, Mismatch of component interfaces, Inconsistent system evolution, Missing/misleading/confusing error information, Incorrect change to counting, Overflow, Improper handling of data structures, Data faults e.g., units, assigned values or problems with the actual input data, invalid input data, Initialisation faults e.g., lack of initialisation of the runtime environment while the program initially executes or restarts, Timing faults
Test (5)	Mismatch of components configurations
Release (6)	Mismatch of components configurations, Configuration management faults e.g., software incompatible with other components
Maintenance (7)	Mismatch of components configurations, change impact faults

Because the reviewed studies did not explicitly specify the software development lifecycle phase in which the defects originated, the classified defect categories were mapped to the IEC

62304 - Medical device software — Software life-cycle processes using the approach proposed by Rajaram *et al.,* (2019). Their work demonstrated that the IEC 62304 lifecycle processes can be aligned with the SW91 main categories as shown in table 2.4. Table 2.5 presents the resulting mapping of the identified defect categories to the corresponding IEC 62304 lifecycle processes.

Table 2.4**:** IEC 62304 and SW91 (Rajaram *et al.,* 2019)

SN	**IEC 62304 Medical Device Software lifecycle processes**	**SW91, Classification of defects in health software**
1	Software development planning	Planning (1)
2	Software requirements analysis	Requirements (2)
3	Software architectural design	Architecture and Design (3)
4	Software detailed design	Architecture and Design (3)
5	Software unit implementation	Implementation (4)
6	Software integration and integration testing	Implementation (4), Test (5)
7	Software system testing	Test (5)
8	Software release for utilization at a system level	Release (6)
9	Software maintenance process	Maintenance (7)

Table 2.5**:** SW91 main defect categories and the identified software defects

SW91 Defect category	**Identified software defects**	**IEC 62304 Medical Device Software lifecycle processes**
Planning (1)		
Requirements (2)	Inconsistent logics from requirements, Incorrect logic in requirements specification, Improper limits, inadequate requirements, Unexpected behaviour of two or more conditions occurring simultaneously, Vital system functions are missing, Lack of documentation or improper documentation, Missing/wrong functions of human-machine interactions, Alarm or message error, Display or image error, Fault tolerance e.g., incorrect action	Software requirements analysis

	due to operator error, Interface faults e.g., software does not properly interface with external device or other software components, Quality assurance faults e.g. the processes were not sufficient, Exceptional conditions were not specified in requirements specification, Function missing in the requirements specification, requirements specification was incorrect for its usage with other components, Test hooks or monitors were not specified, omissions, incorrect facts, inconsistencies, ambiguities, erroneous information, consequences	
Architecture and Design (3)	Block or unblock interrupts fault, Incorrect approximation/precision, Mismatch of reused components, Mismatch of component interfaces, Inappropriate use of keyboard/button, Timing faults, configuration management fault	Software architectural design Software detailed design
Implementation (4)	Exception handling fault, Execution orders of functions fault, Conditional statement fault, Incorrect arithmetic/formula, Incorrect /outdated constants, calculation fault, Incorrect conversion, Mismatch of component interfaces, Inconsistent system evolution, Missing/misleading/confusing error information, Incorrect change to counting, Overflow, Improper handling of data structures, Data faults e.g., units, assigned values or problems with the actual input data, invalid input data, Initialisation faults e.g., lack of initialisation of the runtime environment while the program initially executes or restarts, Timing faults, logic fault, omission fault	Software unit implementation Software integration and integration testing
Test (5)	Mismatch of components configurations, inadequate validation, inadequate software mitigation	Software system testing
Release (6)	Mismatch of components configurations, Configuration management faults e.g.,	Software release for utilization at a system level

	software incompatible with other components	
Maintenance (7)	Mismatch of components configurations, change impact faults	Software maintenance process

2.6 Existing Techniques for Addressing Defects in MDS

The literature reveals various techniques that have been developed and employed to address defects in MDS development. Like in other software projects, in MDS development, testing is used to find defects. MDS manufacturers have employed taxonomy based testing, leveraging generic taxonomies like IBM's Orthogonal Defect Classification (ODC) and HP's Defect Origins, Types, and Modes to accomplish testing (Wagner 2008; Rajaram et al. 2019). However, MDS manufacturers find generic defect-based testing techniques difficult to use due to stringent regulations and the cost and difficulty of building and validating a detailed taxonomy (Rajaram et al. 2019b). AAMI developed AAMI SW91-2018 – Classification of Defects in Health Software in 2018 (ANSI/AAMI 2018), providing a common language for the classification of software defects occurring in health software. SW91, as is commonly called, allows developers to perform a root cause analysis using keywords to determine the defect type that they may have experienced in their product. While defect taxonomies such as SW91 help manufacturers to know defect types, actual detection is still reliant on software testing, typically conducted after significant development work has been done. Similarly, IEC 62304 standard offers general descriptions of MDS development life cycle processes but lacks specific methods on preventing defects. Preventing and identifying defects during development process of MDS is of ultimate importance.

Jetley and Chelf (2009) proposed static analysis for identifying defects in MDS. This non-execution technique of software can be implemented using several ways like formal methods, MISRA standard coding guidelines etc. Tools have also been developed to automate static analysis such as FindBugs and CodeSonar (Johnson et al. 2013; Jetley et al. 2008). Despite its defect detection potential, academic literature discussing static analysis implementation in MDS development is scarce. Johnson et al. (2013) identified several barriers to its adoption by developers that include: 1) inflexibility of static analysis tools to accommodate developer customisations, leading to inability to apply their specific coding styles, 2) high false positives

and false negatives overshadowing true positives, 3) the difficulty in interpreting tool outputs, and 4) inability to check correctness based on the developer's intent.

Hovorushchenko and Popov (2021) proposed the use of Artificial Neural Networks (ANNs) to detect residual defects after basic testing. This approach follows basic testing, which is conducted later in the development process, after significant development work has been completed. This can result in late detection of defects, leading to costly rework (Boehm and Basili 2001). On the other hand, by preventing and identifying defects early in the development process, expensive rework can be avoided. Implementing agile practices during the initial stages of MDS development can prevent defects and avoid costly rework later. The core objective of this research is to address defects in MDS by implementing agile practices to prevent them early on and identify those that may occur later in the development process. This avoids costly rework and potential catastrophes resulting from defects that occur during use.

2.7 Software Development Life Cycle Models

Software Development Life Cycle (SDLC) is an integral part of any software development. The SDLC encompasses detailed steps and activities that describe how to design, develop, and maintain a software (Saravanos and Curinga 2023). In the context of MDS development, the SDLC is particularly crucial as it provides structure, control and assurance that the software adheres to technical and regulatory requirements and fulfils its intended purpose (Swanlund et al. 2024). Consequently, a specific standard, IEC 62304, outlines the lifecycle processes for MDS development. SDLC models can be categorised as either traditional/plan-driven (heavy-weight) or agile (light-weight) methods (Khan et al. 2012). Traditional SDLC models primarily focus on detailed documentation, comprehensive planning, and extensive design (Khan et al. 2012). These models include Waterfall, V-Model, and Spiral Model. Agile SDLC models focus on process adaptability and customer satisfaction through rapid delivery using a mix of iterative and incremental processes (Trivedi 2021).

Agile SDLC emerged in the 1990s as a new approach to flexibly and collaboratively manage software development projects as many were failing or took long to complete (Cohen et al. 2004). These models allow firms to quickly build software solutions while responding to changing business requirements. Examples of light-weight SDLC models include eXtreme Programming (XP), Scrum, Crystal, Adaptive Software Development, and Feature Driven Development (FDD). The choice of a suitable SDLC depends on various factors such as

environment in which the software is developed and its intended purpose (Ruparelia 2010; Nazia 2019). The following subsection presents an overview of various plan-driven SDLC models that could be used in MDS development. Agile methods are discussed in subsection 2.7.2

2.7.1 Plan-Driven SDLC Models

2.7.1.1 Waterfall Model

Since the 1950s, the rapid increase in software demand has made it inevitable for firms to implement a formalised approach to software development. Additionally, the failure of numerous software development projects led most firms to implement an SDLC to enhance their chances of success (Saravanos and Curinga 2023). Among the various SDLC models, the Waterfall model is the oldest. Originally defined by Winston Royce in 1970 (Royce 1987), the key feature of the Waterfall model is that development activities are carried out in linear sequential phases, where one phase is completed before moving to the next (Balaji 2012), as shown in figure 2.1. Therefore, each phase of the Waterfall model, except for the system requirements, is dependent on the deliverables of the preceding one. This model requires that users define a comprehensive set of requirements before the software development process can commence. At the end of each stage of the Waterfall model, documentation is generated that must be approved by the customer before the project can proceed to the next stage (Fruhling et al. 2005).

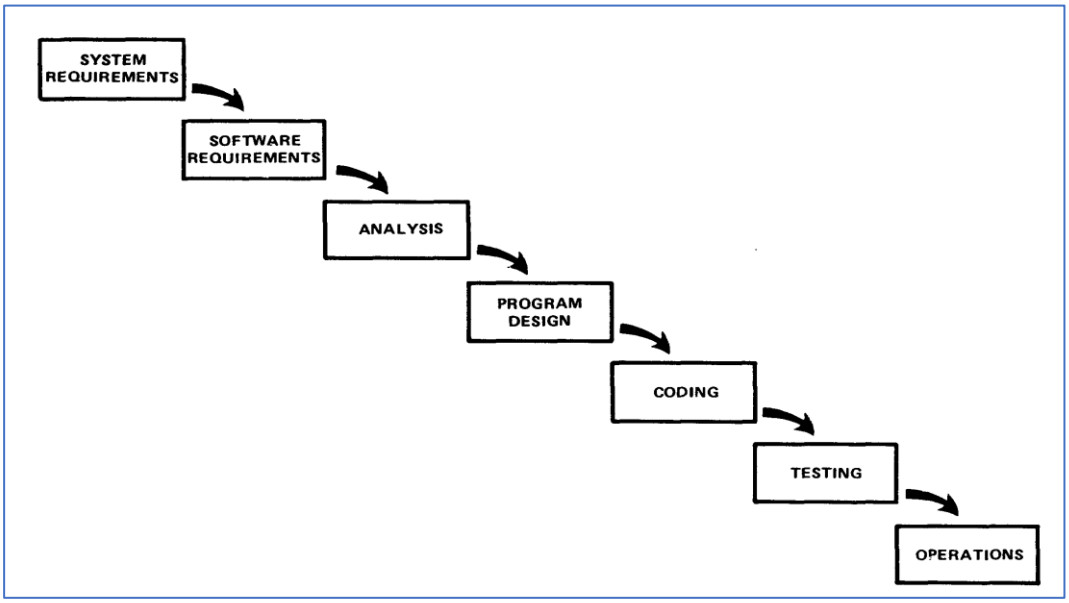


Figure 2.1: Waterfall Model (Royce 1970)

Due to its linear sequential approach, the Waterfall model has received criticism as being inflexible because it does not allow changes to a previous phase without restarting the entire project (Saravanos and Curinga 2023). This in turn can lead to time wastage, project budget overruns, late or suspended deliveries, and ultimately customer dissatisfaction (Bassil 2012; Fruhling et al. 2005). Moreover, when using the Waterfall model, defects are often detected late in the development process (Petersen et al. 2009). The model also presents the risk of developing software that does not meet the needs of the customer or end-users. However, research reveals that the Waterfall model is still commonly used in most software development projects (Saravanos and Curinga 2023). The Waterfall model's prominence is attributed to its clearly defined stages, well understood deliverables, ease of planning tasks and processes, and focus on adequate documentation (Senarath 2021). The model's straightforward structure makes it relatively easy to understand, implement, and execute. While the original Waterfall model defined by Royce includes seven stages, it is also commonly adapted and presented in the literature with five or six stages. For example, Petersen et al. (2009) present it with five stages, while Khan (2023) present an adaptation with six stages. The Waterfall model is ideal for projects with clear requirements and where detailed planning can easily be drafted for the entire project (Despa 2014).

2.7.1.2 The V-Model

The V-Model is considered an extension of the Waterfall model, with the distinction that instead of a linear sequential form, the process steps are bent upwards forming the typical V-shape, with the left and right arm connecting at the lowest level (Mathur and Malik 2010; Johnsson et al. 2023). In the V-Model, the left arm represents analysis activities that organise user requirements into smaller modules. This side begins with system-level requirements, which are divided into subsystems and modules, then specified and implemented in detail (Abraham 2015). The bottom part is the coding activity where the design is translated into code. The right arm of the V-Model represents the verification and validation on the activities carried out on the left side. These activities confirm that the software has been built right and corresponds to the defined requirements (Johnsson et al. 2023). Each stage on the left arm has a corresponding verification on the right arm. The right arm starts with the unit testing stage, moving upwards to the integration testing stage, and finally user acceptance testing stage. A visual representation of the V-Model is provided in figure 2.2.

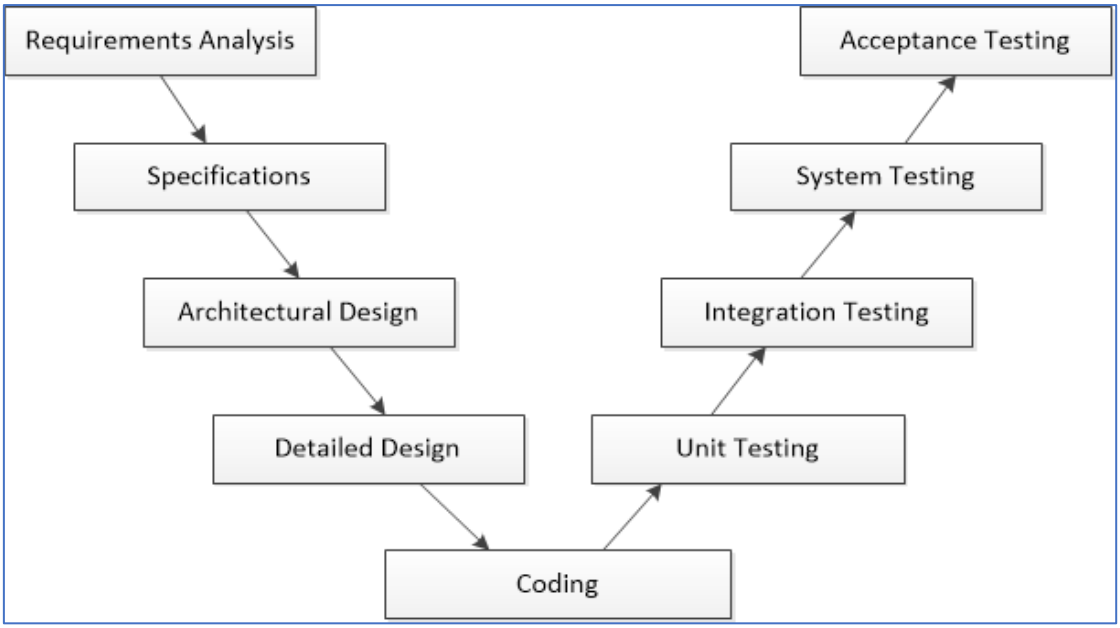


Figure 2.2: The V-Model (Mathur and Malik 2010)

The V-Model is widely used in various domains, including aerospace, automotive, railway, and medical (Durmus et al. 2018; Hauschild et al. 2022). It is recommended for safety standards such as IEC 61508, EN 50126, EN 50128, and IEC 62278 in domains like nuclear, railway, machinery and power drive systems (Durmus et al. 2018). The V-Model is a popular choice for developing safety-critical software because of its capability to produce deliverables that are essential for regulatory approval (Hauschild et al. 2022; Liu et al. 2016; McHugh et al. 2014). An important regulatory aspect, traceability, is inherently supported in the V-Model, which facilitates clear linkage between requirements, implementation and testing (McCaffery et al. 2012). Additionally, the V-Model's emphasis on planning for verification and validation activities in the early stages of development makes it particularly suitable for safety-critical software development (Sonigara et al. 2023). Furthermore, the V-Model's phase-oriented structure with specific deliverables and review processes at each phase, contributes to its manageability and effectiveness in ensuring development rigor (McHugh, Cawley, et al. 2013).

Various adaptations of the V-Model are presented for diverse purposes across software development domains. For example, Mathur and Malik (2010) present the Advanced V-Model which incorporates testing and maintenance activities. Durmus et al. (2018) proposed the Enhanced V-Model by extending the detailed design stage to provide a final inspection of modules in relation to faults before proceeding to coding and module testing phases in safety-

critical systems development. Liu et al. (2016) present an adaptation called an Incremental V-Model (inc-V), specifying it for safety-critical systems development in the automotive domain.

In the context of MDS development, research reveals that the V-Model is widely used (Hauschild et al. 2022; Johnsson et al. 2023; Mardiah et al. 2022; Özçelik and Altilar 2020; McHugh et al. 2013). Additionally, as discussed later in this chapter, the V-Model has been widely adapted by integrating agile practices within it for various purposes in MDS development. Despite its wide use in MDS development, the V-Model does not provide mechanisms for preventing defects. Instead, testing is typically done towards the end of the development lifecycle where identifying and fixing defects can be very costly (Durmus et al. 2018).

The V-Model shares similar disadvantages as the Waterfall model in that it is rigid and adjusting project scope is difficult and costly (Munassar and Govardhan 2010). Additionally, the V-Model does not provide a clear path for problems discovered during the testing phases (Munassar and Govardhan 2010). Despite its shortcomings, the V-Model provides capabilities to meet the requirements for regulatory compliance and is the common approach in MDS development (Hauschild et al. 2022).

2.7.1.3 The Spiral Model

The Spiral model was developed based on the waterfall model to provide a risk-driven approach to software development (Boehm 1988). It was created to address the shortcomings of the waterfall model such as inflexibility and long development time as mentioned in section 2.5.1.1. The Spiral model combines elements of both the waterfall and prototyping models, involving iterative rounds and activities that start from the center and proceed outwards. Unlike the Waterfall model, the Spiral model does not require detailing and defining the entire system at the outset. The Spiral model comprises four iterative phases: requirements definition, risk analysis, project implementation, and evaluation. Each iteration in the spiral model follows a phase-wise linear process. After the first iteration, the customer evaluates the software and gives feedback (Tanwar 2016). This feedback is incorporated into the succeeding iteration, which again follows the linear approach to implement the suggested improvements. The spiral model is shown in figure 2.3.

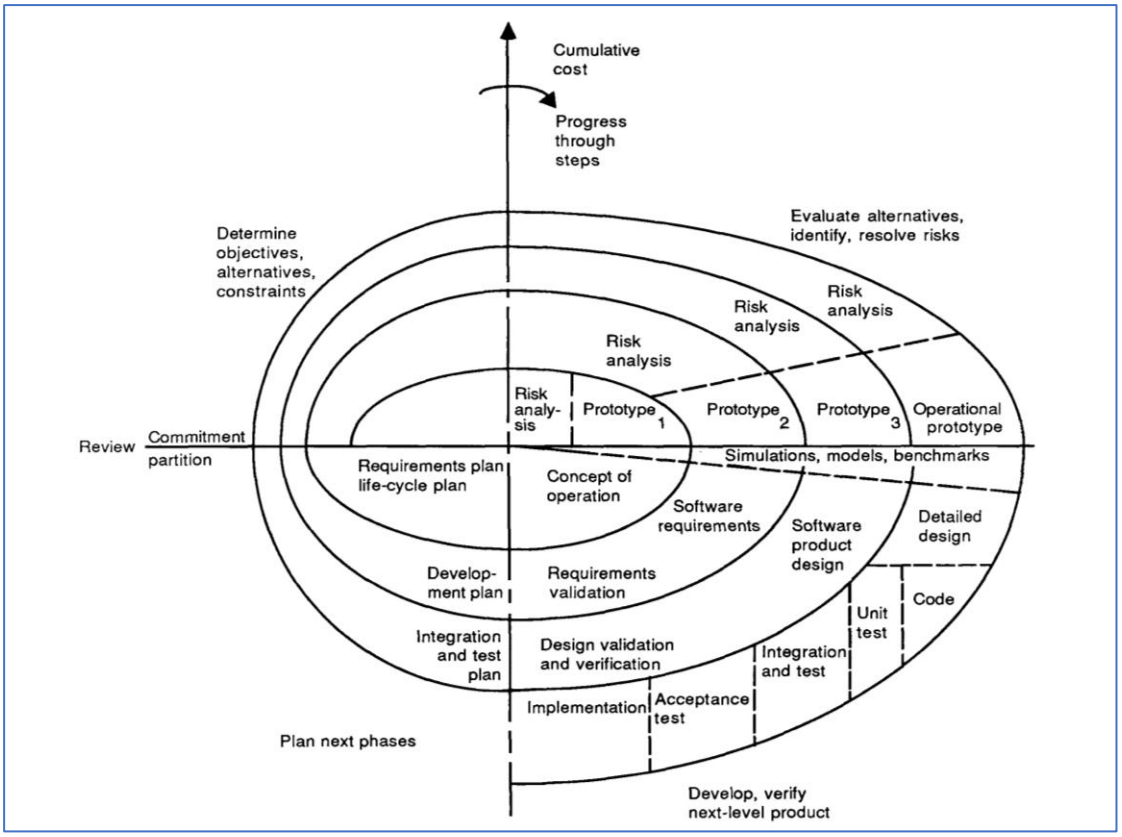


Figure 2.3: The Spiral model (Boehm 1988)

The key advantages of the spiral model include its focus on risk analysis and early customer feedback based on the developed prototypes (Yu 2018). The spiral model is particularly suitable for large and mission-critical projects (Munassar and Govardhan 2010). However, the model is complex and requires several experts experienced in risk analysis (Yu 2018). Additionally, the numerous iterations can make the model expensive to implement and may lead to significant delays in project completion (Yu 2018).

2.7.2 Agile Software Development

ASD emerged in 2001 as an alternative to traditional or plan-driven methods like Waterfall (Highsmith 2002). ASD was created to address the limitations of plan-driven methods. It reduces overhead and costs while affording the flexibility to accommodate changing requirements at any stage (Alsaqqa et al. 2020). ASD reduces the risk of developing low quality software by minimizing defects through lightweight methods that emphasise customer collaboration, early delivery of functional pieces of software which are continuously improved, and responsiveness to change (Noor and Fahad Khan 2014; Abdelaziz et al. 2015; Beecham et al. 2014; Turk et al. 2005; Diebold 2013). ASD shortens delivery time while meeting changing

client needs through fast feedback and flexibility to accommodate rapid changes to requirements. Abrahamsson et al. (2002) describe a key aspect of ASD as the implementation of lightweight but adequate project behaviour rules along with human and communication-oriented guidelines.

There are many ASD methodologies or models such as Scrum and eXtreme Programming (XP) (Ibrahim 2007; Abrahamsson et al. 2002; Highsmith 2002). Each ASD methodology has several practices. For example, Scrum has practices such as sprint, sprint backlog, and sprint planning (Sutherland and Sutherland 2014), and XP has practices such as refactoring, pair programming and collective code ownership (Abrahamsson et al. 2002; Beck 2000). Agile practices are used to ensure principles and values of an agile methodology are implemented (F. Tripp and Armstrong 2018). The next subsection provides an overview of agile methodologies.

2.7.2.1 Agile Methodologies

Agile methodologies are founded on the four values and 12 principles outlined in the Agile Manifesto for software development published by software practitioners and consultants (Beck et al. 2001). The four values of the agile manifesto are stated as follows:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

These values indicate that agile methodologies emphasise on teamwork and customer feedback for the success of a software development project, in contrast to heavy-weight SDLCs which focus on extensive documentation. The 12 principles of the Agile Manifesto are designed to guide teams on how to execute software projects with agility and achieve high-value capabilities (Marbach et al. 2015). The 12 principles of the Agile Manifesto as outlined by Martin (2014) are:

1. *Our highest priority is to satisfy the customer through early and continuous delivery of valuable software*. This principle emphasises the importance of development teams delivering a limited functionality system within the first few weeks of the project to obtain customer feedback and then continue to deliver increasingly functional systems.

2. *Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage*. This principle underscores the importance of agile development teams embracing requirements changes and viewing such changes as catalysts for developing a high-quality software that meets customer needs. Therefore, agile development teams ensure their software structure is flexible to accommodate requirements changes with minimal impact.

3. *Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale*. This principle means that the primary goal of agile development teams should be the early delivery of working software that meets customer needs, rather than focusing on extensive documentation or plans.

4. *Businesspeople and developers must work together daily throughout the project*. Martin (2014) elaborates that agile projects should emphasise frequent interactions between customers and the development team. This ensures that customer needs are met through continuous feedback and necessary improvements.

5. *Build projects around motivated individuals. Give them the environment and support they need and trust them to get the job done*. Agile projects value people as the key success factor for any project. Therefore, incentivising teams and creating a conducive working environment are important for the successful execution of software projects using lightweight SDLCs.

6. *The most efficient and effective method of conveying information to and within a development team is face-to-face conversation*. This principle means that in agile projects, direct communication or talking to each other is the primary mode of conveying information.

7. *Working software is the primary measure of progress*. Martin (2014) explains that in agile projects, progress is measured primarily by the extent to which the software meets customer needs.

8. *Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely*. Agile projects are run at a fast, but sustainable pace, avoiding team exhaustion and ensuring the highest quality standards throughout the project's duration. Sustainable pace is a rate at which the development team can consistently work comfortably over an extended period (Rubin 2012).

9. *Continuous attention to technical excellence and good design enhances agility*. The author explains that agile development teams prioritise correcting errors as soon as they are discovered, ensuring continuous production of high-quality software.
10. *Simplicity - the art of maximizing the amount of work not done--is essential*. The author elaborates that agile teams focus on building the simplest software that permits modifications to address any issues that may arise later.
11. *The best architectures, requirements, and designs emerge from self-organizing teams*. In agile projects, responsibilities are communicated to the entire team, which then determines the best way to fulfil them.
12. *At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly*. This principle means that agile teams should be flexible and adapt to their environment to remain relevant.

In addition to the 12 principles, Abrahamsson et al. (2002) identify nine characteristics of agile methodologies that facilitate faster delivery and shorten project lifecycles:

1. Modularity on development process level
2. Iterative with short cycles enabling fast verifications and corrections
3. Time-bound with iteration cycles from one to six weeks
4. Parsimony in development process removes all unnecessary activities
5. Adaptive with possible emergent new risks
6. Incremental process approach that allows functioning application building in small steps
7. Convergent (and incremental) approach minimizes the risks
8. People-oriented, i.e. agile processes favour people over processes and technology
9. Collaborative and communicative working style

The following subsections provide detailed overviews of various agile methodologies.

2.7.2.1.1 Scrum

Scrum is a lightweight SDLC developed as an enhancement of the iterative and incremental object-oriented development approach (Schwaber 1997). Initially, it was designed as a

methodology for managing, enhancing, and maintaining existing systems or prototypes (Schwaber 1997). Scrum defines three key roles within a Scrum team: Scrum Master, Product Owner, and Development Team (Adkins and Smith 2016; Noll et al. 2017). The Scrum Master facilitates scrum ceremonies and manages communications between the Product Owner and the team. Additionally, the Scrum Master plays a leadership role in removing impediments that hinder team progress (Rubin 2012). The Product Owner represents product stakeholders or users and is responsible for managing the product backlog (Schwaber and Sutherland 2020). This role involves deciding which features and functionalities to build and determining the order of implementing them. Tasks related to managing product backlog include:

1. Developing and explicitly communicating the Product Goal
2. Creating and clearly communicating Product Backlog items
3. Ordering Product Backlog items
4. Ensuring that the Product Backlog is transparent, visible and understood.

The Development Team is self-organised, cross-functional, self-managing, and is responsible for delivering a potentially shippable increment of working software (Schwaber and Sutherland 2020). Therefore, it is important that the development team must collectively poses all the skills required to produce high-quality working software (Rubin 2012).

Scrum defines six ceremonies: Product Backlog Refinement, Daily Scrum, Sprint, Sprint Planning, Sprint Review, and Sprint Retrospective (Noll et al. 2017; Schwaber and Sutherland 2020). Each of these ceremonies is an opportunity to inspect and adapt Scrum artefacts (Schwaber and Sutherland 2020). Sprints, typically lasting one month or less, are key to achieving value from ideas in Scrum implementation. They are timeboxed and each Sprint is supposed to produce something of tangible value to the customer (Rubin 2012). Sprint Planning initiates the Sprint by outlining the most important tasks to be completed, with entire Scrum team collaborating to create this plan. The Daily Scrum is typically a 15-minute ceremony focused on inspecting progress towards the Sprint goal, adapting the sprint backlog as required and adjusting the scheduled work (Schwaber and Sutherland 2020). During the Daily Scrum, each team member provides an update about what they have done, what they are doing, and what they plan to do. The Sprint Review is a timeboxed ceremony focused on assessing the Sprint's resulting product increment and determining necessary future adaptations. The Sprint Retrospective focuses on assessing the previous Sprint and identifying ways to enhance quality

and effectiveness (Schwaber and Sutherland 2020). Product Backlog Refinement is a continuous ceremony throughout the Scrum project. It involves the following (Scrum Alliance 2024):

1. Confirming the order of the product backlog items
2. Removing or demoting items that no longer seem important
3. Adding or promoting items that come up or become more important
4. Splitting larger items into smaller items
5. Merging smaller items into larger items
6. Estimating items
7. Identifying which items are sprint-ready

Scrum also defines four artefacts: Product Backlog, Burndown Chart, Sprint Backlog, and Product Increment (Lopes de Souza et al. 2021). The Product Backlog is a prioritised list of ideas for the product that the Development Team commits to completing during a particular sprint. It may also include changes to existing features, defects to be fixed, and technical improvements (Rubin 2012). The Burndown Chart is used to monitor the progress of the Sprint and to determine when items should be removed from the Sprint Backlog for the next Sprint. The Sprint Backlog specifies how the team plans to design, build, integrate, and test the selected features from the Product Backlog during a specific sprint. The Product Increment is an important deliverable of every Sprint. It must be of high-quality, usable, meet the Development Team's definition of done, and acceptable to the Product Owner (Scrum Alliance 2024). The definition of done consists of a checklist of tasks that the team must complete before declaring their work potentially shippable (Rubin 2012). The Scrum model is shown in figure 2.4.

Another important aspect of Scrum is requirements management. In Scrum, requirements details are continuously negotiated through conversations during development and are refined just-in-time for the Development Team to start building the required functionality (Rubin 2012). The handling of these requirements is accomplished through user stories which provide a convenient format for representing items of business value for the product's features. To convey the important detail about a feature to the Scrum team and stakeholders, user stories must be simple and unambiguous. A common template format for writing user stories and an example user story are shown in figure 2.5.

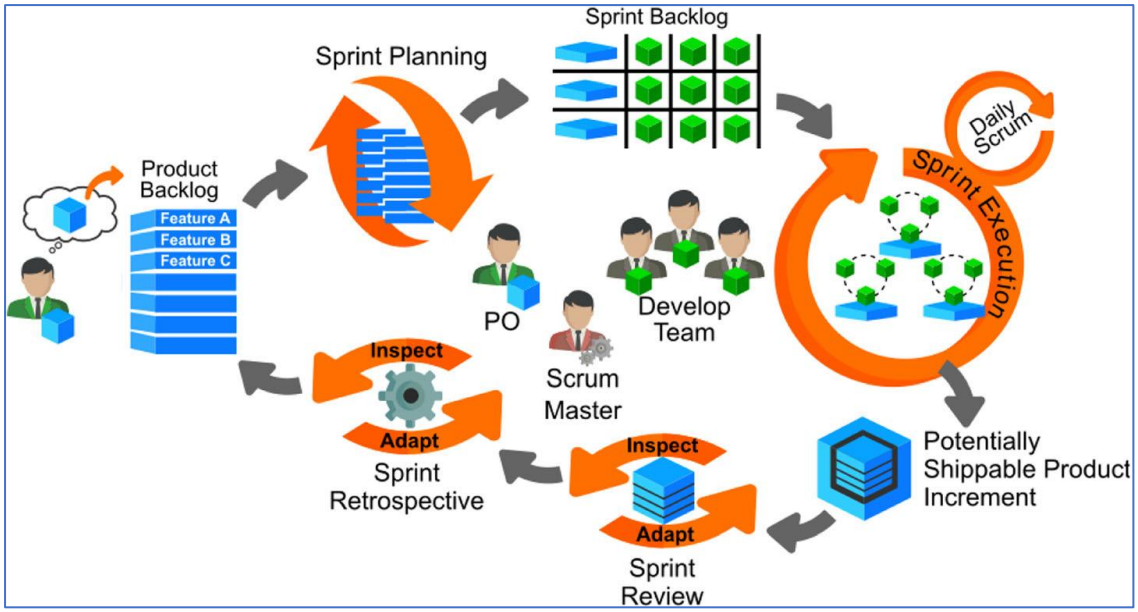


Figure 2.4: Scrum Model (Lopes de Souza et al. 2021)

A user story consists of three parts: card, conversation, and confirmation (Rubin 2012). The card serves as a placeholder with limited space to capture the intent of a requirement. The conversation is the actual communication among the Scrum team and stakeholders to flesh out requirements details from a user story. The confirmation includes acceptance criteria for the desired functionality of a feature. It helps the development team understanding what to build and test to meet customer needs and allows the Product Owner to judge if the user story has been satisfactorily implemented.

In the literature, Scrum ceremonies and artefacts and user story are also collectively referred to as “practices” such as in the work of Abrahamsson et al. (2002). This research adopts the term practices to encompass these elements.

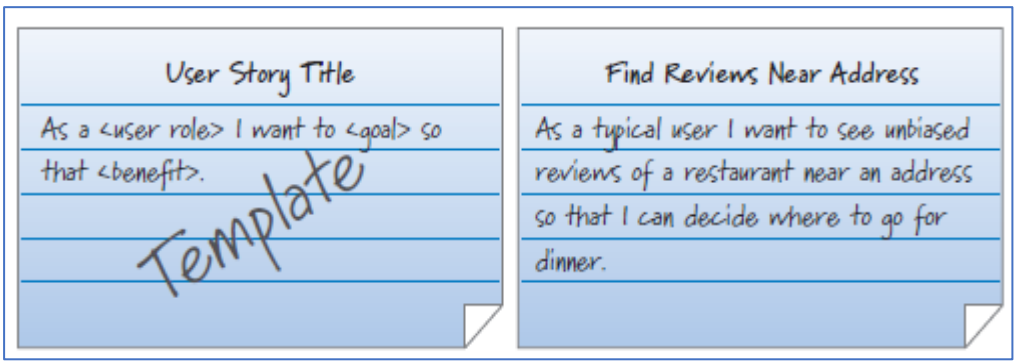


Figure 2.5: User Story template and example User Story on card (Rubin 2012)

2.7.2.1.2 eXtreme Programming

eXtreme Programming (XP), developed by Kent Beck in 1996, is based on five core values: communication, simplicity, feedback, courage, and respect (Beck and Andres 2004). XP emphasises the excellent application of programming techniques, clear communication, and teamwork. Beck recognised that software development is a team activity that relies on active communication to disseminate information among team members. Consequently, XP emphasises face-to-face communication as the optimal means for information exchange within the development team and the customer. Simplicity means keeping things such as system architecture as basic as possible to facilitate easy maintenance, adaptability, and waste reduction. Systems should be as simple as possible, but they must work. Feedback between the customer and the development team is crucial as it enables the development team to make necessary adjustments to the product based on the customer requirements. This feedback loop contributes to quality by ensuring that any mistakes are corrected in a timely manner. Courage relates to the development team's ability to abandon ineffective methods and their eagerness to explore new methods to bring about change. The development team should have the courage to alter the structure of their code whenever necessary. XP emphasises mutual respect among team members and encourages constructive criticism that strengthens the team to achieve valuable solutions.

Beck and Andres (2004) explain that XP focuses on addressing risks at all levels of software development and producing high-quality software. While XP was originally developed for small-medium teams (Smith and Stoecklin 2001), Beck, in the second edition of his book acknowledges that it can work with teams of any size (Beck and Andres 2004). There are several characteristics that distinguish XP from other SDLCs (Beck and Andres 2004):

- Its short development cycles, resulting in early, concrete, and continuing feedback.
- Its incremental planning approach, which quickly comes up with an overall plan that is expected to evolve through the life of the project.
- Its ability to flexibly schedule the implementation of functionality, responding to changing business needs.
- Its reliance on automated tests written by programmers, customers, and testers to monitor the progress of development, to allow the system to evolve, and to catch defects early.

- Its reliance on oral communication, tests, and source code to communicate system structure and intent.
- Its reliance on an evolutionary design process that lasts as long as the system lasts.
- Its reliance on the close collaboration of actively engaged individuals with ordinary talent.
- Its reliance on practices that work with both the short-term instincts of the team members and the long-term interests of the project.

The XP model is shown in figure 2.6.

In addition to its core values, Beck founded XP on 12 primary practices that enable software development teams to appropriately meet these values (Lindstrom and Jeffries 2004). These practices are planning game, simple design, test-driven development, coding standard, refactoring, pair programming, collective code ownership, continuous integration, small releases, system metaphor, onsite customer, and 40-hour week. Later, additional practices were introduced in Beck's second edition of his book, increasing the number from 12 to 34. Beck and Andres (2004) categorised them into primary (providing improvement independently) and corollary (providing improvement based on mastery of primary ones) practices as shown in table 2.6.

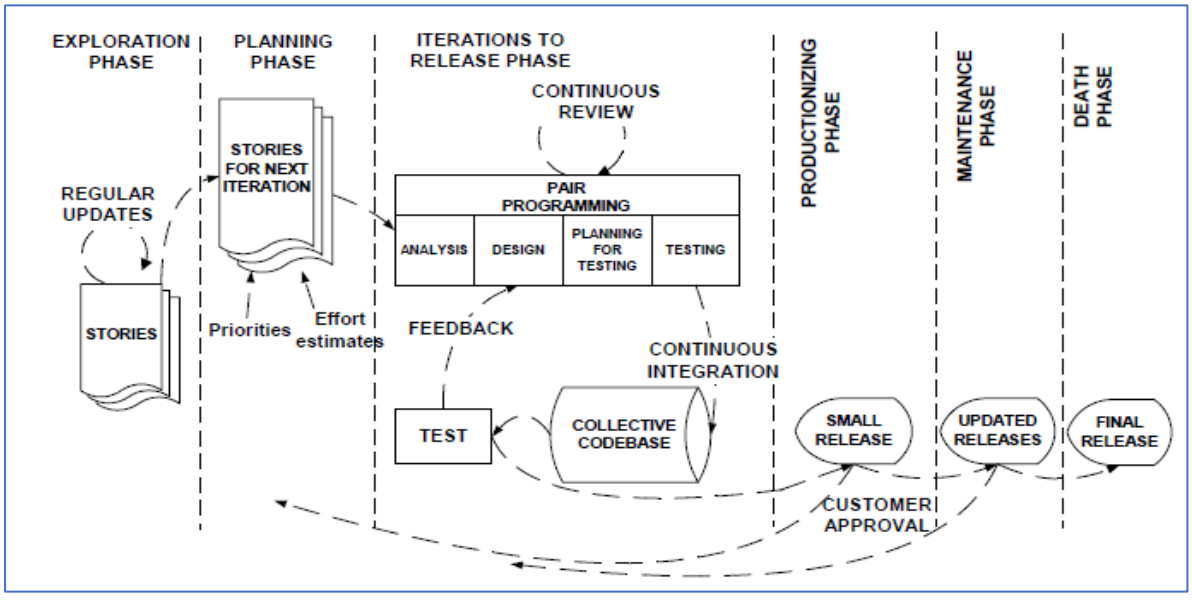


Figure 2.6: XP Model (Abrahamsson et al. 2002)

Table 2.6**:** Primary and Corollary XP practices

Primary Practices	**Corollary Practices**
Sit Together	Real Customer Involvement
Whole Team	Incremental Deployment
Informative Workspace	Team Continuity
Energized Work	Shrinking Teams
Pair Programming	Root-Cause Analysis
Stories/ User Stories	Shared Code
Weekly Cycle	Code and Tests
Quarterly Cycle	Single Code Base
Slack	Daily Deployment
Ten-Minute Build	Negotiated Scope Contract
Continuous Integration	Pay-Per-Use
Test-First Programming	
Incremental Design	

Wake (2002) depicts the XP model as an onion consisting of three layers, each representing different purposes for the 12 core practices, as shown in figure 2.7. Lindstrom and Jeffries (2004) refer to a similar concept as the "Circle of Life", highlighting importance of these core practices when executing XP projects. The inner layer represents programming activities typically performed in pairs. The practices related to this layer include simple design, testing, refactoring, and coding standards. The middle layer encompasses team-oriented practices such as collective ownership, continuous integration, metaphor, coding standards, 40-hour week, pair programming, and small releases. The outer layer represents the processes by which the team interacts with the customer, including practices like onsite customer, acceptance testing, small releases, and planning game.

Despite the introduction of additional practices in Beck's second edition, literature such as Darwish (2011), Kumar and Dwivedi (2021), Geambaşu et al. (2011), and Yadav and Yasvi (2019) indicates that the most widely adopted and used practices remain the original 12 core practices defined in the first edition. XP uses a practice similar to Daily Scrum known as Daily Standup Meeting where each team member describes what they are working on, progress, what they found out, and any problems encountered (Dudziak 2000).

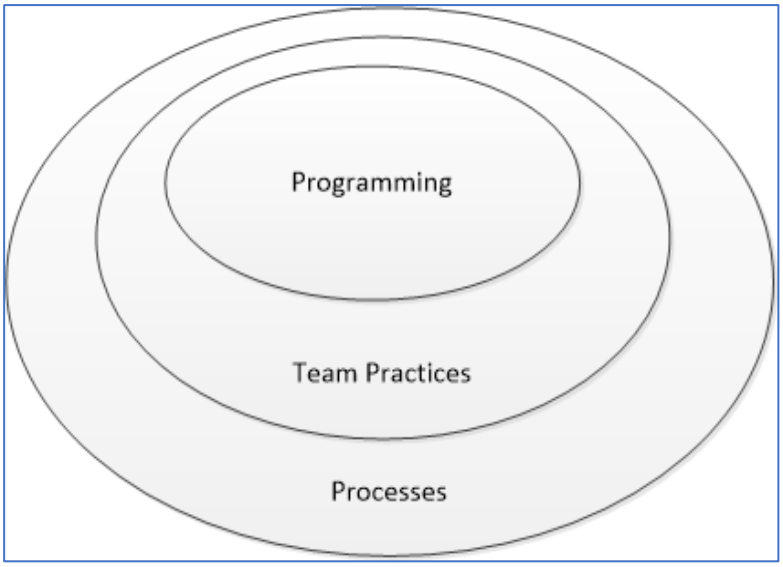


Figure 2.7: The XP onion (Wake 2002)

The following subsections present an overview of each of these practices of XP (Beck and Andres 2004).

1. Planning Game

This practice comprises three steps: exploration, planning and steering (Dudziak 2000). The exploration step involves the customer defining the desired system features using user stories, and developers estimating the schedule to implement them. For a better estimation, developers can create a spike solution - a very simple program to explore potential solutions - for ambiguous user stories (Dalalah 2014). The planning step involves negotiations between the customer and developers to determine which features can be accomplished within the determined schedule. In the steering step, the negotiated schedule is implemented and updated as necessary, based on lessons learned during development and changes in the business environment. This step consists of four elements: iteration, recovery, new story, and re-estimation (Dudziak 2000). Iterations are the key components of the development process. Recovery involves renegotiating the schedule, which can result in changes to the release date or scope. A new story can be added by the customer based on changes in the business environment, necessitating a renegotiation of the schedule. Re-estimation occurs when the development team deems the plan to be inaccurate and requires renegotiation.

2. Onsite Customer

XP projects are customer-driven, requiring the real customer to always be available to provide requirements, set priorities, answer questions, and provide feedback which steers the project

(Dalalah 2014). However, literature reveals that getting first-hand access to the real customer can sometimes be difficult. As a result a knowledgeable customer proxy is often used in XP projects (Lohan et al. 2011). In some instances, the project manager can serve as the customer proxy (Inayat et al. 2012).

3. Small Releases

XP emphasises that a simple, yet functional system should be deployed quickly, followed by frequent short cycle releases of new versions with added features (Paige et al. 2005). These small releases are important as they provide end users the opportunity to evaluate the system and give feedback to the development team for continuous improvements. However, achieving frequent releases that satisfy the customer during each iteration can be challenging for the development team and can lead to significant overhead (Dalalah 2014).

4. System Metaphor

This practice enables the project team to describe the overall system in an unambiguous manner to achieve a shared vision that guides the team towards a valuable solution (Paige et al. 2005).

5. Simple Design

During the project design stage, XP encourages teams to keep the design as simple as possible ensuring any complexity is removed as soon as it is discovered (Paige et al. 2005). This practice can significantly reduce the time wasted on developing solutions that do not work (Dalalah 2014).

6. Refactoring

This practice involves developers restructuring the system without changing its behaviour to reduce duplication, simplify the design, and improve communication (Paige et al. 2005). Refactoring should be followed by adequate testing to ensure the system continues to function as required (Dalalah 2014; Dudziak 2000).

7. Coding Standards

This practice encourages developers to write code by adhering to set rules and emphasising clear communication through the code (Paige et al. 2005). Implementing this practice ensures consistency within the system's code despite contributions from multiple programmers (Dalalah 2014).

8. Pair Programming

XP emphasises that all system code should be written by two programmers at one machine (Paige et al. 2005) or, in modern contexts, through remote collaborative IDEs such as Live

Share, CodeTogether, and Replit. By having Programmers work in pairs, errors can easily be detected, problematic code can be easily identified and refactored, and both programmers can benefit from instant feedback on their ideas. This practice ultimately leads to high quality code and facilitates knowledge sharing among programmers thereby improving their skills as they learn from each other (Dalalah 2014).

9. Collective Ownership

This practice involves each member of the development team being able to make changes to any part of the code, thereby avoiding bottlenecks that arise due to reliance on only specific members for code refactoring (Dalalah 2014).

10. Continuous Integration

This practice involves the development team rapidly building and integrating code daily whenever a feature is completed (Paige et al. 2005). Through implementation of this practice, compatibility issues can be resolved early rather than later where they can be costly to resolve (Dalalah 2014).

11. 40-hour week

XP encourages teams to adhere to the rule to work no more than 40 hours per week and no overtime to maintain efficiency (Paige et al. 2005; Dalalah 2014).

12. Test-Driven Development

This practice was initially called Testing in Beck's first edition of the book (Beck 2000). It was renamed 'Test-First Programming' in the second edition (Beck and Andres 2004). Beck formalised this practice, which is now commonly called Test-Driven Development (TDD) in the literature. TDD involves planning and writing tests before development. Implementing this practice results in high-quality code that is easier to maintain and significantly reduces the amount of testing required (Rust et al. 2007). An important aspect of implementing TDD in XP projects is the verification and validation of the tests within the programming pairs (Paige et al. 2005). This requires that tests written by one member of the pair be verified and validated by the other member.

2.7.2.1.3 Crystal Methodology

Crystal is a family of development approaches created by Alistair Cockburn in the 1990s to address communication problems faced by development teams (Cohen et al. 2004). These approaches can be selected and used based on the nature of the project (size, complexity, and criticality) and available resources (Abrahamsson et al. 2002). The different Crystal approaches

are depicted using different colours ranging from light to darkest, indicating decreasing agility (Cohen et al. 2004). Therefore, the most agile among the Crystal approaches is Crystal Clear, followed by Crystal Yellow, Crystal Orange, Crystal Red, etc. as shown in figure 2.8. As the number of people in the project increases, the right-side approaches become more suitable. Similarly, the project's increasing criticality necessitates movement upwards on the graph. These elements are represented by a combination of a letter and a number in figure 2.8. The letters indicate the criticality level as follows: Comfort (C), Discretionary money (D), Essential money (E), and Life (L) (Abrahamsson et al. 2002). Comfort (C) relates to a situation where a defect causes a loss of comfort, while Life (L) relates to a situation where a defect can cause loss of life. For example, L20 in figure 2.8 refers to a project carried out by a maximum of 20 people to develop a system with the highest criticality level, Life.

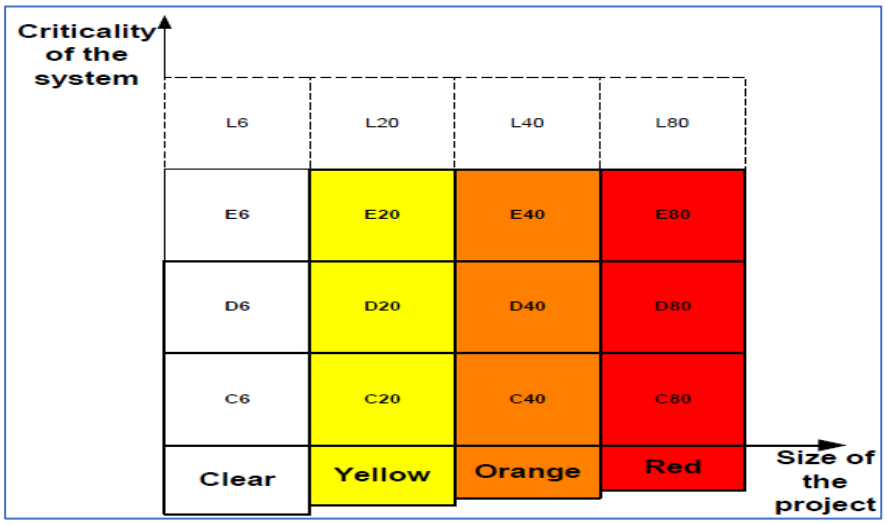


Figure 2.8: Family of Crystal Approaches (Abrahamsson et al. 2002)

2.7.2.1.3.1 Characteristics of Crystal Methodology

Crystal defines seven properties that teams adhere to for successful project execution (Cockburn 2005):

1. Frequent delivery

This property emphasises that teams implementing Crystal approaches deliver working versions of the system to real users every few months. This allows the users to test the system and assess whether their requirements are being met. Additionally, this property also helps the development team collect feedback from users and make any necessary changes which ultimately leads to a high-quality final product. This property is similar to the practice of small releases in XP.

2. Reflective improvement

This feature focuses on preventing project failure by ensuring that the development team meets for at least an hour every few weeks to assess progress on the project and discuss necessary improvements. During these meetings, the team identifies successes, impediments, discusses possible improvements, and agrees on the necessary changes to implement.

3. Osmotic communication

This attribute emphasises face-to-face collaboration among the development team members enabling them better to answer pertinent questions and discuss project aspects as a team. In such arrangements, communication flows naturally and team members filter out unnecessary information and focus only on what is important. However, osmotic communication can be challenging for larger teams due to increased noise and numerous questions requiring the team's expert developer to address them.

4. Personal safety

This property focuses on creating an environment that encourages team members to clearly communicate their concerns about the project to both the project leadership and other team members without fear of being mistreated.

5. Focus

This attribute emphasises on clearly communicating the project's goal and having team members quickly identify, schedule, and implement the necessary tasks.

6. Easy access to expert users

This feature suggests that expert users should be readily available to the development team throughout the duration of the project to test the working versions of the system, provide necessary feedback, and requirements.

7. Technical Environment with Automated Tests, Configuration Management & Frequent Integration

This property emphasises teams to implement automated testing and select an appropriate configuration that allows frequent integration and delivery as well as easy rollback in the event of a failure.

2.7.2.1.3.2 Crystal Methodology Practices

Similar to Scrum and XP, several practices have been defined that apply to the Crystal family of development approaches (Abrahamsson et al. 2002). The following subsections provide an overview of these practices.

1. Staging

This practice involves planning and scheduling the delivery of a working increment every month or within a maximum of four months throughout the project's duration (Abrahamsson et al. 2002). This practice is similar to the practice of small releases in XP.

2. Revision and Review

This practice focuses on development teams assessing progress through activities that take place within each iteration which include construction, demonstration, and reviewing the objectives of the increment (Abrahamsson et al. 2002).

3. Monitoring

This practice suggests that deliverables are the key determinants of the team's progress and stability. Progress is assessed through the team's milestones and stability stages (Abrahamsson et al. 2002). Milestones include start, review 1, review 2, test, and deliver, while stability stages comprise wildly fluctuating, fluctuating, and stable enough to review (Abrahamsson et al. 2002).

4. Parallelism and Flux

This practice allows two tasks to be carried out simultaneously when the development team deems the deliverables 'stable enough to review' during monitoring (Abrahamsson et al. 2002). This enables different teams to work on multiple tasks in parallel.

5. Holistic diversity strategy

This practice emphasises creating multiple smaller cross-functional teams with diverse specialties (Abrahamsson et al. 2002). However, special consideration should be given to team location, communication, documentation, and coordination, as these issues may impact the project.

6. Methodology-tuning technique

This practice provides the development team with the flexibility to adjust the development process based on lessons learned during the development of an increment (Abrahamsson et al. 2002). These adjustments can be implemented during the development of the next increment.

7. User Viewings

Crystal emphasises that users should assess each delivered increment of the system twice for smaller projects and thrice or more for larger projects (Abrahamsson et al. 2002). This reduces the time wasted in the actual testing of the product.

8. Reflection Workshops

Crystal encourages development teams to conduct reflection workshops both before and after every increment with the aim of allowing teams to learn from the feedback and experiences (Abrahamsson et al. 2002).

2.7.2.1.4 Feature-Driven Development

Feature-Driven Development (FDD) was founded by Jeff De Luca and Peter Code in the 1990s and was first introduced in their book "Java Modelling in Colour with UML" (Chowdhury and Huda 2011). In 2001, Stephen Palmer and Mac Felsing published a book specifically focused on FDD, titled "A Practical Guide to Feature-Driven Development" (Palmer and Felsing 2001). FDD focuses on the design and development of features rather than the entire development process (Abrahamsson et al. 2002). The design aspect involves collaborating with the client to identify a list of features that will be implemented (Chowdhury and Huda 2011). This is an important part as the success of the project primarily depends on the quality of work done during this stage. The features included in the features list should be small enough to be developed in short iterations, typically of maximum two weeks (Palmer and Felsing 2001). Additionally, features should be unambiguous to help developers produce something that is valuable to the client. Hunt (2006) distinguishes a feature from user requirements, user stories, and use cases in the context of FDD by defining it as a "schedulable piece of functionality" that delivers value to the user. Additionally, features are associated with the following (Hunt 2006):

- a priority (so that they can be ordered)
- a cost (so that they can be accounted for)
- resources (so that they can be scheduled)

The development part in FDD involves taking each feature from the list and implementing it within a short iteration. In other words, the features are implemented in a feature-by-feature fashion. The FDD process is shown in figure 2.9.

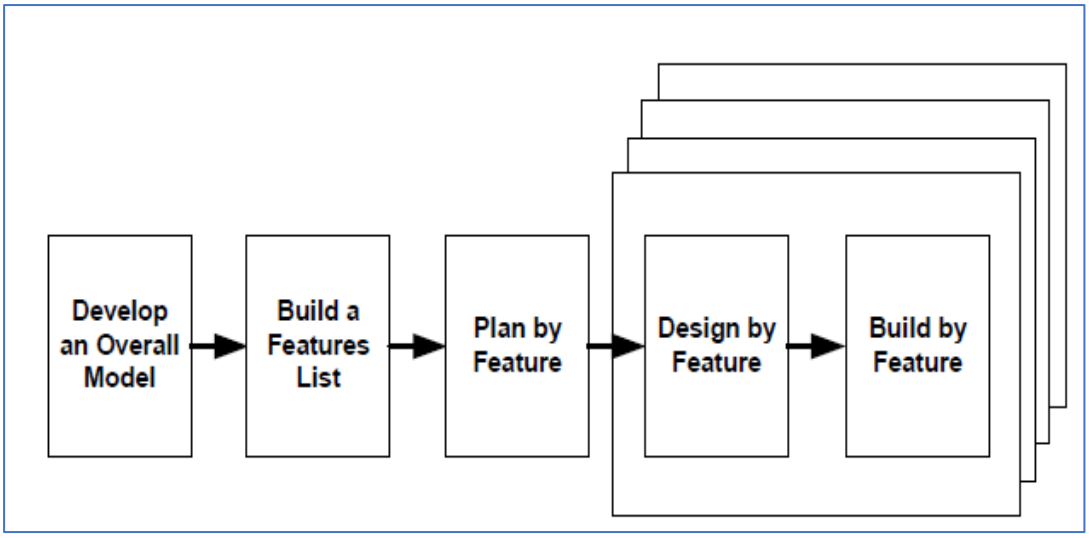


Figure 2.9: FDD process (Abrahamsson et al. 2002)

2.7.2.1.4.1 FDD Practices

FDD is founded on a set of eight 'best practices' that its creators advocate to be implemented collectively as they are more beneficial when used together (Abrahamsson et al. 2002). An overview on each of the practices is presented next.

1. Domain Object Modeling

This practice involves developing class diagrams to represent the types of objects and their relationships within the problem domain. Additionally, sequence diagrams are used to illustrate how objects interact with each other to fulfil their responsibilities. Therefore, the primary focus in this practice is to remove ambiguities in requirements and establish a common understanding of the problem domain (Palmer and Felsing 2001). The overall result of this practice is a framework to which features are added (Abrahamsson et al. 2002). This framework guides teams in producing improved initial designs without requiring significant refactoring of the classes to include new features (Palmer and Felsing 2001).

2. Developing by Feature

Based on the successful completion of the previous practice, this practice involves implementing each of the classes in turn, ensuring that each one adds value to the client's desired system functionality (Palmer and Felsing 2001; Abrahamsson et al. 2002).

3. Individual Class (Code) Ownership

Unlike XP, FDD emphasises single ownership of a class or piece of code within the development team. This means that each developer is assigned a class or a set of classes for

which they are responsible for conceptual integrity, consistency, and performance (Palmer and Felsing 2001; Abrahamsson et al. 2002).

4. Feature Teams

Feature teams are small dynamically formed development teams with members drawn from class owners as needed (Palmer and Felsing 2001; Abrahamsson et al. 2002). They are responsible for implementing features in a feature-by-feature fashion (Palmer and Felsing 2001).

5. Inspections

Inspections are reviews or evaluations performed on designs and code. Inspections are important in FDD for ensuring high-quality designs and code (Palmer and Felsing 2001). They also help in defect detection, knowledge transfer, and standards conformance (Palmer and Felsing 2001).

6. Regular Builds

This practice involves building a working system at regular intervals such as weekly or daily, by integrating all completed features and allowing the client to assess its value and provide feedback (Palmer and Felsing 2001).

7. Configuration Management

A configuration management is used for identifying source code for completed features and maintaining a history of changes made to the classes (Palmer and Felsing 2001).

8. Reporting or Visibility of Results

This practice involves frequently and accurately tracking and reporting project progress to all roles within and outside the project (Palmer and Felsing 2001).

2.7.2.1.5 Dynamic Systems Development Method

Dynamic Systems Development Method (DSDM) was created in 1995 to "provide a framework of controls and best practices for Rapid Application Development (RAD)" (Hunt 2006). DSDM emphasises flexibility in requirements changes while keeping the project timeline and resources fixed (Abrahamsson et al. 2002). The goal of DSDM is to produce high-quality systems rapidly at lower costs (Hunt 2006). DSDM is founded on nine principles (Hunt 2006), also referred to as practices (Abrahamsson et al. 2002), as follows:

1. Active user involvement is imperative. Key users that understand what is expected of the system should be present throughout the project to provide feedback.

2. The team must be empowered to make decisions. The team should be able to make timely and accurate decisions to steer the project in the right direction.
3. The focus is on frequent delivery of products. This enables users to provide feedback on the system increments and developers to make necessary corrections based on the feedback.
4. Fitness for business purpose is the essential criterion for acceptance of deliverables. Developers should ensure to deliver solutions that are of business value to clients.
5. Iterative and incremental development is necessary to converge on an accurate business solution. This ensures that defects are detected and corrected early.
6. All changes during development are reversible. This provides the team the flexibility to abandon incorrect paths and revert to previous correct states, ultimately leading to a quality product.
7. Requirements are baselined at a high level. This provides the flexibility for detailed requirements to be changed while core requirements are frozen at a high level. This allows development to start even with fewer requirements. Additionally, as development progresses more requirements are frozen as they become clearer and agreed upon (Abrahamsson et al. 2002).
8. Testing is integrated throughout the life cycle. This ensures that each increment is tested by both users and developers rather than waiting until all components are developed.
9. Collaboration and cooperation between all stakeholders is essential. This is important for the smooth running of the project, ultimately contributing to its success.

2.7.2.1.5.1 DSDM Phases

The DSDM model comprises seven different phases: Pre-Project, Feasibility Study, Business Case Study, Functional Model Iteration (FMI), Design and Build Iteration (DBI), Implementation, and Post-Project Phase (Hunt 2006). The Pre-Project, Feasibility Study, and Business Case Study phases are done sequentially only once. These phases are important as they enable users and the development team to understand the environment in which the system will operate and the final system itself. The FDI, DBI, and Implementation phases are the core phases in DSDM where actual development work is done. These phases are iterative and incremental. The FMI phase involves three activities: 1) analysis of the features to be designed and implemented, 2) production of the Functional Model, and 3) coding and prototyping. The

DBI phase focuses on designing and building the features and writing code to implement the required functionality. The Implementation phase involves deployment of the completed system into the actual production environment. This phase also includes user training and development of user manuals. The Post-Project phase involves activities such as help desk support to ensure the deployed system operates effectively and ascertaining that the expected business benefits are realized. The DSDM model is shown in figure 2.10.

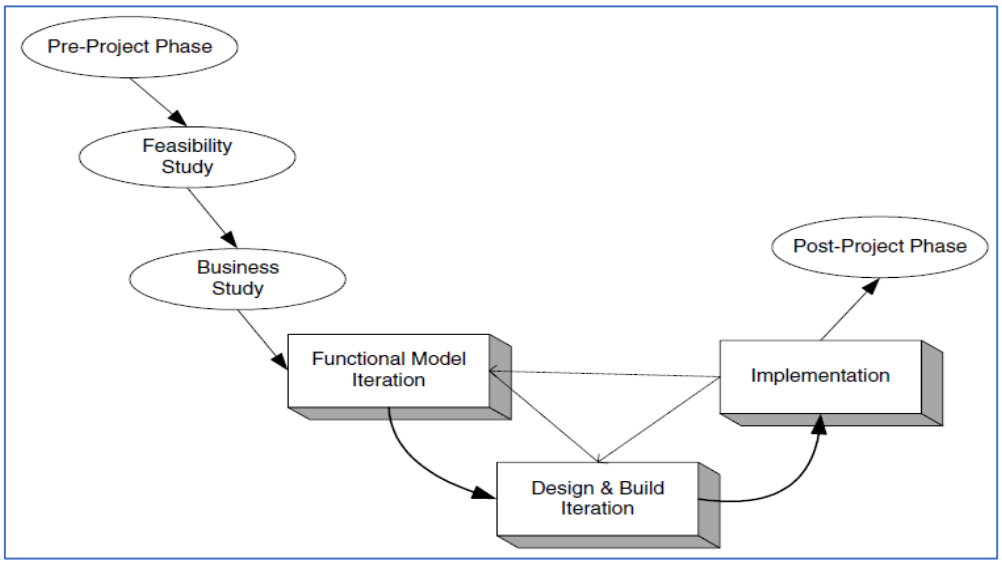


Figure 2.10: DSDM model (Hunt 2006)

2.7.2.1.6 Adaptive Software Development

Adaptive Software Development was created by James A. Highsmith III in the early 1990s and first published in the year 2000 (Orr 2001). Adaptive Software Development was developed to address challenges in developing complex large-scale systems, by providing a framework and guidance to prevent systems projects failures (Abrahamsson et al. 2002). The method emphasises continuous adaptation, collaboration, rigor, and discipline while encouraging creativity in executing large and complex projects (Highsmith 2000). Unlike the plan-design-build lifecycle inherent in plan-driven approaches, Adaptive Software Development provides a flexible lifecycle through three main phases: speculate, collaborate, and learn (Orr 2001).

The speculate phase focuses on short iterations intended for exploration and experimentation, embracing uncertainty rather than viewing it as a weakness, and avoiding the perception of deviation from a plan as failure (Orr 2001; Abrahamsson et al. 2002). Consequently, in Adaptive Software Development, results, identified as features (parts of the product that bring value to the customer) to be developed during an iteration, are important than tasks or actions

(smallest unit of work). The collaborate phase stresses the importance of teamwork as a means to exploit diverse knowledge and experiences in developing high-quality systems (Orr 2001). This phase involves short iteration of design-build-test cycles. The learn phase emphasises continuous learning through retrospectives, customer focus groups and reviews of every iteration. The activities in this phase can be likened to the sprint review and retrospectives in Scrum. The Adaptive Software Development lifecycle is illustrated in figure 2.11.

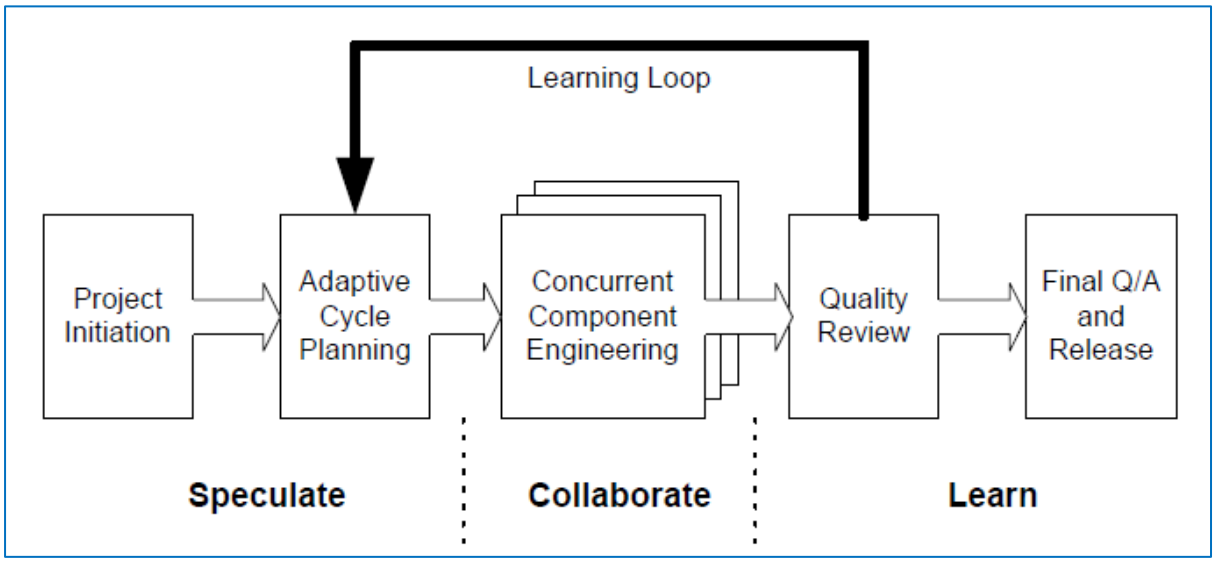


Figure 2.11: Adaptive Software Development lifecycle (Abrahamsson et al. 2002)

2.7.2.1.6.1 Adaptive Software Development Practices

Adaptive Software Development defines three main practices: iterative development, feature-based planning, and customer focus group reviews (Abrahamsson et al. 2002). This limited number of practices may be attributed to Adaptive Software Development's focus on results rather than tasks as previously mentioned. Iterative development is a core practice in Adaptive Software Development that facilitates improvement and adaption. This practice involves breaking the development process into smaller, short iterations, that encompass planning, designing, developing, testing and obtaining customer feedback. Feature-based planning suggests Adaptive Software Development's emphasis on delivering customer functionality during an iteration. The practice of customer focus group reviews indicates Adaptive Software Development's commitment to continuous feedback which helps development teams to adapt and improve.

2.7.2.1.7 Rational Unified Process

The Rational Unified Process (RUP), created by Philippe Kruchten and colleagues at Rational Software Corporation in the 1990s, has evolved and is currently maintained by IBM's Rational Software (Anwar 2014; Péraire et al. 2007). RUP aims to provide best practices and guidance for the effective use of the Unified Modeling Language (UML), an industry-standard modeling method that enables development teams to unambiguously communicate requirements, designs, and architectures (Abrahamsson et al. 2002; Anwar 2014). RUP emphasises the creation of models, particularly use cases, that clearly represent the software to be developed. The goal of RUP is to enable development teams to deliver high-quality systems that meet user needs within the project's defined timeframe and budget (Péraire et al. 2007).

RUP defines four iterative phases for software projects: Inception, Elaboration, Construction and Transition (Abrahamsson et al. 2002; Péraire et al. 2007). The inception phase involves assessing the project's viability, costs, and risks. An agreement is formed among all stakeholders on whether the project is worth pursuing or not. The activities during the inception phase include (Péraire et al. 2007):

1. Formulating the scope of the project. This involves capturing the context and the most important requirements and constraints to such an extent that you can derive acceptance criteria for the end product.
2. Planning and preparing a business case. Evaluating alternatives for risk management, staffing, the project plan, and cost, schedule, and profitability trade-offs.
3. Synthesising a candidate architecture, evaluating trade-offs in design, and in make, buy, and reuse, so that cost, schedule, and resources can be estimated. The goal is to demonstrate feasibility through a proof of concept by using a model or an initial prototype, to identify high-risk areas of the project.
4. Preparing the environment for the project, assessing the project and the organisation, selecting tools, and deciding which parts of the process to improve.

The elaboration phase involves establishing a stable system architecture for the main design and implementation in the next phase. The activities in this phase include (Péraire et al. 2007):

1. Defining, validating, and baselining the architecture as rapidly as practical.

2. Refining the vision, based on new information obtained during the phase, establishing a solid understanding of the most critical use cases that drive the architectural and planning decisions.
3. Creating and baselining detailed iteration plans for the Construction Phase.
4. Refining the development process and putting in place the development environment, including the process, tools, and automation support required to support the construction team.
5. Refining the architecture and selecting components. The selected architectural components are integrated and assessed against the primary scenarios.

The construction phase involves identifying remaining requirements and actual development of the system. The activities carried out during this phase include (Péraire et al. 2007):

1. Resource management, control, and process optimisation
2. Complete component development and testing against the defined evaluation criteria
3. Assessment of product releases against acceptance criteria for the vision

The transition phase involves iterations of testing, adjusting, and feedback to ensure the software is delivered for use by its intended users. Activities during this phase include (Péraire et al. 2007):

1. Executing deployment plans
2. Finalising user support material
3. Testing the deliverable product at the development site
4. Creating a product release
5. Getting user feedback
6. Fine-tuning the product based on feedback
7. Making the product available to users

The RUP lifecycle process is presented in figure 2.12.

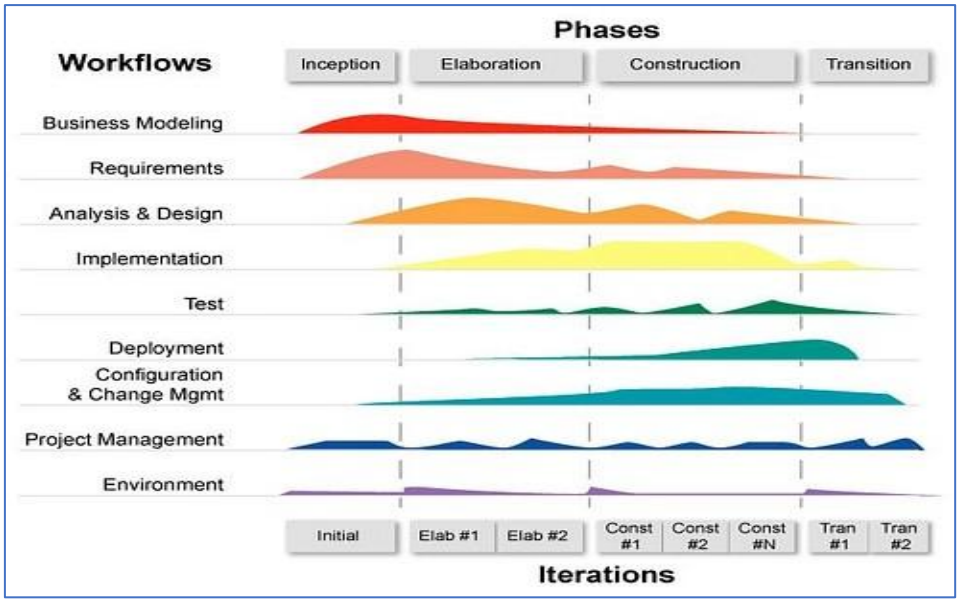


Figure 2.12: RUP lifecycle (Kruchten 2000)

2.7.2.1.7.1 RUP Practices

RUP defines six primary practices for software development teams: develop software iteratively, manage requirements, use component-based architectures, visually model software, verify software quality, and control changes to software (Abrahamsson et al. 2002; Kruchten 2000). The following subsections provide an overview of each of these practices.

1. Develop Software Iteratively

This practice emphasises building software in small, short iterations that enable teams to identify potential risks early through working releases and make successive adjustments and refinements based on user feedback.

2. Manage Requirements

This practice focuses on using use cases and scenarios to accurately capture and organise functional requirements. These requirements drive the design, implementation, and testing of the software, ensuring that it meets user needs.

3. Use Component-based Architectures

This practice focuses on early creation of a robust executable architecture before committing resources for full-scale execution of the project. It involves designing a flexible and clear architecture that facilitates software reuse.

4. Visually Model Software

This practice encourages the use of models to help development teams clearly understand both the problem and its viable solution. Models also assist development teams to understand large, complex systems that otherwise would not be easy to comprehend.

5. Verify Software Quality

This practice emphasises that development teams must continuously review software quality throughout the development cycle with respect to requirements based on reliability, functionality, application performance and system performance.

6. Control Changes to Software

This practice focuses on accommodating, controlling, tracking, and monitoring requirements changes while enabling successful iterative development. It also involves controlling changes to software artefacts such as models, code, and documents.

2.7.2.1.8 Agile Model Driven Development

Agile Model Driven Development (AMDD) blends agile development with Model-Driven Development to leverage the benefits of both approaches (Matinnejad 2011). AMDD emphasises the use of agile models, which are defined as "just barely good enough for the situation at hand" (Ambler 2007). Examples of agile models include hand-drawn sketches and data models created using UML or other CASE tools. Generally, agile models possess the following attributes (Ambler 2007):

- Agile models fulfil their purpose.
- Agile models are understandable.
- Agile models are sufficiently accurate.
- Agile models are sufficiently consistent.
- Agile models are sufficiently detailed.
- Agile models provide positive value.
- Agile models are as simple as possible.

AMDD defines a few practices but inherits several from Agile Modeling. AMDD also inherits TDD from XP and adapts Refactoring into Code Refactoring and Database Refactoring. While code refactoring involves modifying code while preserving its behaviour, database refactoring involves modification to the database schema to improve its design while preserving its behaviour and informational semantics. A list of Agile Modeling core practices is shown in Table 2.7.

Table 2.7**:** Agile Modeling Practices

Agile Modeling Practices	
Core Practices	**Supplementary Practices**
Active Stakeholder Participation	Apply modeling standards
Apply the Right Artifact(s)	Apply patterns gently
Collective Ownership	Discard temporary models
Single Source Information	Formalize contract models
Create Several Models in Parallel	Update only when it hurts
Create Simple Content	
Depict Models Simply	
Display Models Publicly	
Iterate To Another Artifact	
Model in Small Increments	
Model With Others	
Prove it With Code	
Use the Simplest Tools	

The AMDD lifecycle encompasses four primary activities (Ambler 2008):

1. Envisioning

During this activity the development team identifies the system's scope and develops its architecture. This activity is typically carried out within the first weeks of the project. Depending on the size of the project, envisioning can last from a few hours to two weeks.

2. Iteration Modelling

This activity involves planning the work to be done during an iteration and using modeling to accurately estimate the effort required for each requirement.

3. Model Storming

Ambler (2008) defines Model Storming as a "just in time modeling" where an issue discovered by a team member is promptly resolved either with a colleague or collaboratively by the entire team. Once a solution is identified, each team member proceeds with their own work.

4. Test-Driven development

In AMDD, TDD helps the development team to validate the software according to the needs of stakeholders and to specify the software on a “just in time” basis. By using TDD, development teams can “capture detailed specifications in form of executable tests instead of static documents or models” (Ambler 2008).

A graphical representation of the AMDD is shown in figure 2.13.

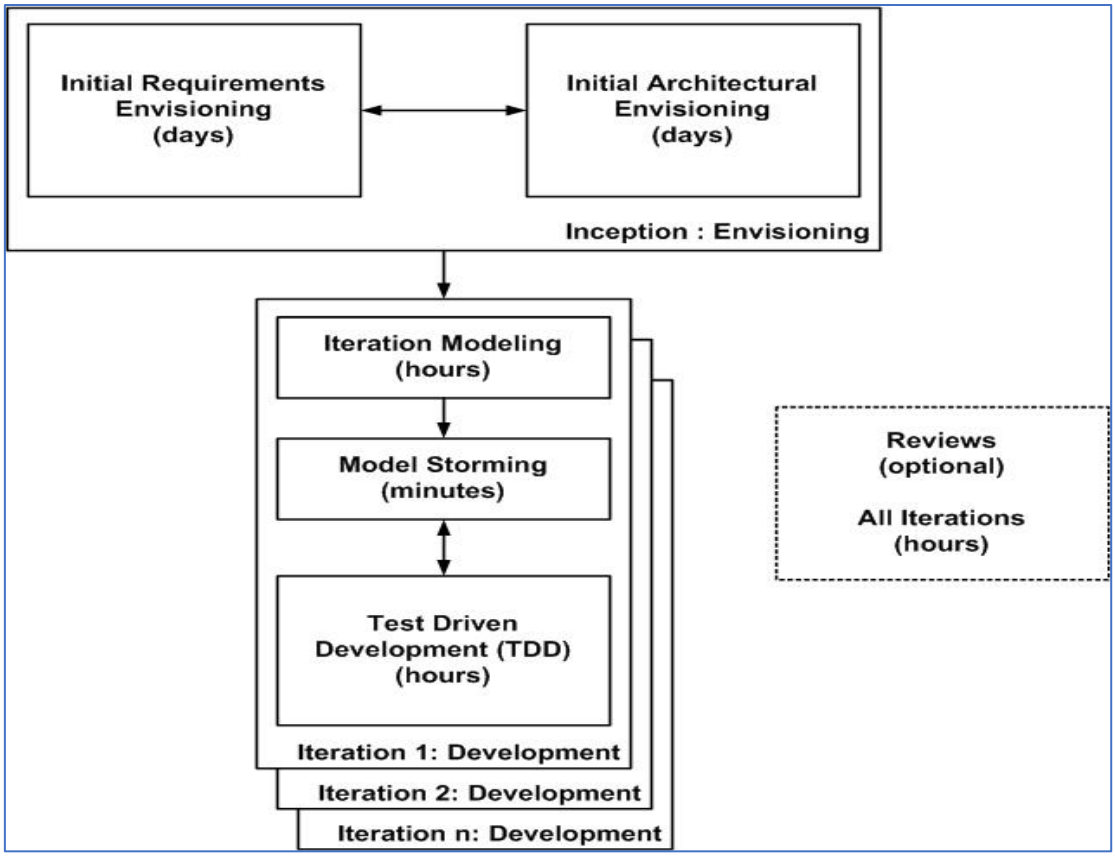


Figure 2.13: AMDD Lifecycle (Ambler 2007)

2.7.2.1.9 DevOps

DevOps is a software development methodology that emphasises integration of software development and IT operations personnel, fostering collaboration and communication (Gokarna and Singh 2021). The collaboration aspect of DevOps improves team performance by eliminating silos between development and operations (Azad and Hyrynsalmi 2023). Additionally, communication is a key element in ASD and significantly contributes to the success of any team. DevOps was formally developed by Patrick Debois in 2009 as a result of applying principles from various domains including physical manufacturing, leadership, and

IT (Kim et al. 2016). Debois coined the term “DevOps” from the DevOPsDays, a conference he created and first organised in Belgium. The term DevOps is derived from the combination of two words: development (Dev) and operations (Ops).

The DevOps community advocates for a unified and cross-functional team to accelerate development while ensuring quality and reliability of deliverables. This team is responsible for development, testing, and operations. These are accomplished through activities such as continuous integration, deployment, monitoring, collaboration, and communication, aimed at automating the development process (Tatineni 2021). DevOps primarily focuses on continuous delivery of the software and continuous integration (Gokarna and Singh 2021).

2.7.2.1.9.1 DevOps Lifecycle

DevOps encompasses eight iterative phases that individually represent activities and techniques for development. The phases include planning, coding, building, testing, releasing, deploying, operating, and monitoring/feedback. These phases are organised into left loop (Dev) and right loop (Ops) indicating different operational aspects, activities and tools (Tatineni 2021; Azad and Hyrynsalmi 2023). The planning phase involves the cross-functional team collaboratively defining project objectives, features, and determining their priorities. This phase also involves organising tasks into smaller manageable units. During coding phase, developers write code for the defined features and implement version control system to track code changes. Additionally, code reviews are carried out during this phase to ensure code quality and conformance to standards.

The build phase involves using automated tools to compile, test, and package code into useful artefacts that are added to the project. These artefacts are then committed into the code base through continuous integration which ensures the project is rebuilt to accommodate the new code. During the test phase, the software artefacts are checked for standards compliance through automation testing. Additionally, this phase involves trial versions of the software being experimented for validation by a small group of users. The release phase involves performing final checks that verifies the software is ready for deployment. In the deployment phase, the software is put into production and checked for any necessary changes. The operation phase involves configuration and management of software applications in the production environment. This phase is followed by monitoring phase in which the deployed

applications are assessed for performance and any identified issues initiates an iteration to improve the software.

The DevOps lifecycle is shown in figure 2.14.

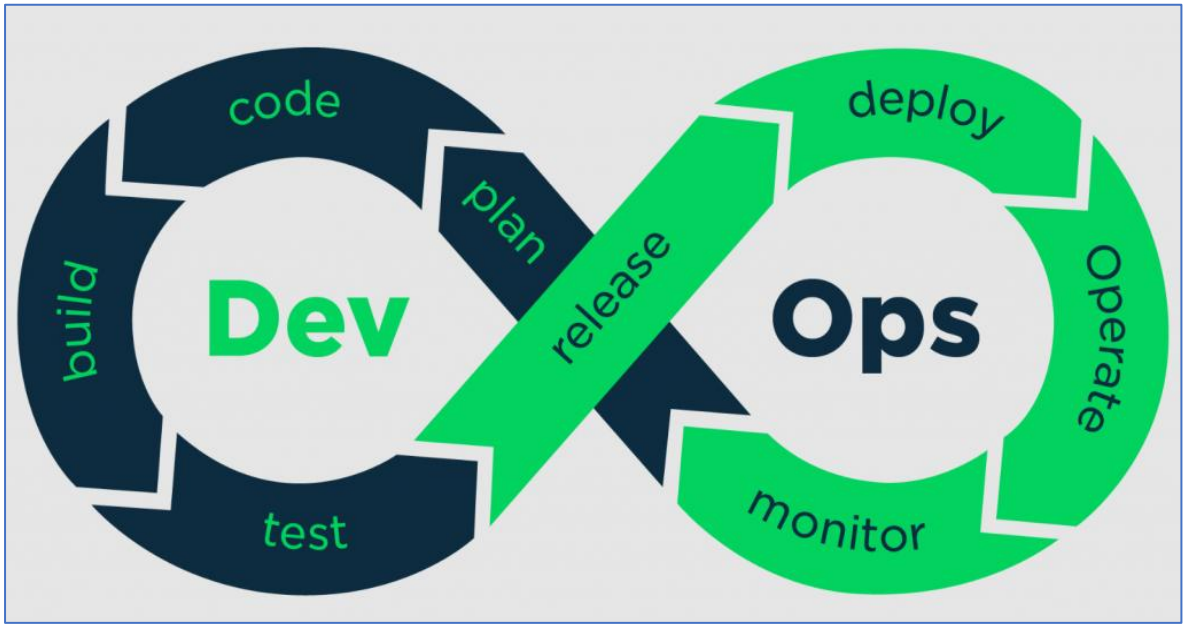


Figure 2.14: DevOps Lifecycle

2.7.2.1.9.2 DevOps Practices

The fundamental principle of DevOps is continuity, evident in its seven key practices: continuous delivery, continuous deployment, continuous development, continuous testing, continuous integration, continuous operation, and continuous monitoring (Azad and Hyrynsalmi 2023). An overview of each practice is provided next.

1. Continuous Delivery

This practice encompasses automatically deploying code to a testing environment which enables the team to release new software features or updates frequently.

2. Continuous deployment

This practice involves the automated and frequent release of software updates to the production environment. The practice aims to make new software always available to users.

3. Continuous development

This practice encompasses ongoing, iterative process of creating and improving software features. It enables a steady flow of functionalities and enhancements.

4. Continuous testing

This practice involves implementing automated testing throughout the development lifecycle on a continuous basis. The aim of this practice is to prevent code changes from negatively affecting the existing functionalities.

5. Continuous integration

This practice involves merging code changes from multiple contributors to a common code base at a regular basis. The primary purpose of this practice is to detect and resolve integration issues early.

6. Continuous monitoring

This practice involves real-time assessment of the software in its production environment to promptly identify issues and enhance performance.

7. Continuous operation

This practice encompasses monitoring the software in its production environment and responding to any incidents. The practice facilitates availability, and reliability of the software in the production environment.

2.7.3 Agile Scaling Frameworks

To build upon the successes of agile methodologies in small-scale projects and address issues such as coordination and integration in multi-team environments within large organisations and large-scale projects, several agile scaling frameworks have been developed (Uludağ et al. 2021). These frameworks include the Scaled Agile Framework (SAFe), Large Scale Scrum (LeSS), and Disciplined Agile Delivery (DAD) (Putta et al. 2018).

SAFe, created by Dean Leffingwell and first released in 2011, focuses on program management and integrates elements from various agile methods such as Scrum, XP, Lean, and DevOps (Mucambe et al. 2020). DAD was developed by Scott W. Ambler and Mark Lines while working at IBM in 2012, to provide a coherent, end-to-end strategy that encompasses both operations and support for agile solutions delivery (Ambler and Lines 2013). LeSS was created by Craig Larman and Bas Vodde in 2005 to extend Scrum to multiple teams collaborating on a large-scale project in multi-site and off-shore environments (Larman and Vodde 2017).

Among the various agile scaling frameworks, research indicates that SAFe is the most commonly adopted (Digital.ai 2023). However, despite its popularity, existing literature suggests that organisations often find SAFe complex to implement (Putta et al. 2018). As a

result, organisations opt to integrate only a subset of its practices into their existing SDLCs (Laanti and Kettunen 2019). The SAFe framework is shown in figure 2.15.

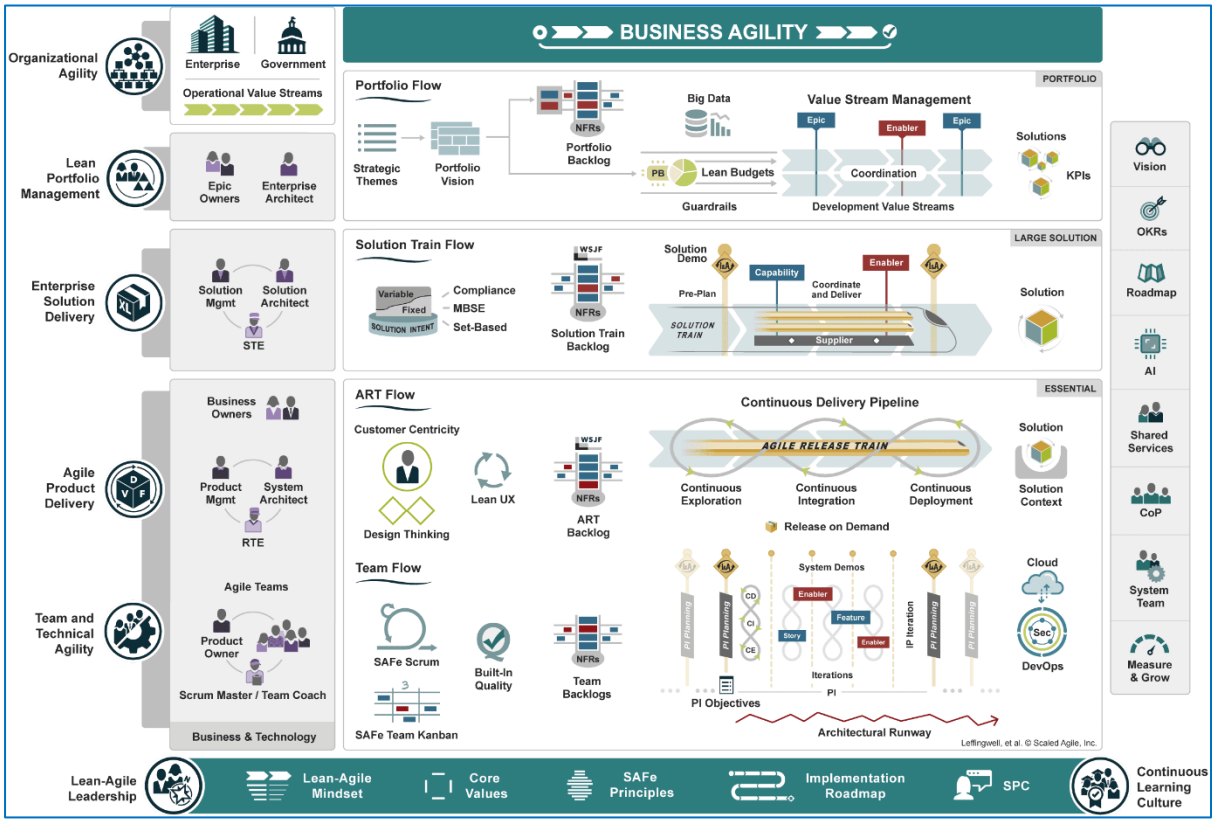


Figure 2.15: SAFe (Mucambe et al. 2020)

2.7.4 Lean Software Development

Lean Software Development (LSD), introduced by Tom and Mary Poppendieck in 2003, is an ASD approach derived from Lean Manufacturing, which itself stems from the Toyota Production System (Middleton and Joyce 2012). LSD focuses on delivering value to the customer quickly. It is based on seven principles (Poppendieck and Poppendieck 2010):

1. *Eliminate waste* – waste in software development refers to anything such as a feature or process that does not bring value to the customer. This includes unnecessary functionalities, delays, work in progress, and unneeded processes. The key element in this principle is ensuring that development teams focus on identifying the precise needs of the customer and delivering only what is required as quickly as possible, rather than developing more features that are not needed.

2. *Amplify learning* – this principle emphasises the importance of continuous learning through feedback provided by the customer after each iteration. The feedback is key to

improving the product by ensuring it aligns with user needs and optimising the development environment.

3. *Decide as late as possible* – this principle pertains to development teams postponing decisions until sufficient information is available. This reduces the likelihood of making choices early in the development process based on speculation rather than facts. Instead, teams should embrace flexibility, incorporating a capacity for change as new insights emerge in their development processes.
4. *Deliver as fast as possible* – this principle emphasises the importance of reducing cycle time and providing the customer with a working system as soon as possible while it is still relevant. Development teams should proactively avoid slack in the development process as it provides the capacity to change, reinventing itself, leading to waste of resources. While accommodating change is important in ASD, development teams should implement a streamlined workflow to ensure timely delivery of valuable solutions.
5. *Empower the team* – this principle focuses on providing development teams with the necessary expertise and involving them in technical decisions. It is important to create an environment in which team members can actively participate and improve themselves. This enables team members to make meaningful contributions to the project. Additionally, communication among team members is important for clarifying what needs to be done for a task.
6. *Build integrity in* – this principle focuses on two key aspects of integrity in software development: perceived and conceptual. Perceived integrity is about the customers long-term perception of the software product. To maintain this, the development team must prioritise customer values through regular customer feedback. Conceptual integrity is about ensuring the system's core components are working together smoothly and cohesively. Additionally, teams must ensure that the software remains useful over time and include the capacity for its smooth evolution, adaptation, and future scalability.
7. *See the whole* – this principle emphasises the importance of team members to focus on the overall system in the project rather than only on their areas of specialisation. For example, designers and modelers should produce models that convey all the business

processes, rules, and other essential elements in a language that every team member can understand.

2.8 Agile in Medical Device Software Development

Agile methodologies present benefits such as reducing overhead and costs while affording the flexibility to accommodate changing requirements at any stage (Alsaqqa et al. 2020). However, research indicates that using these on their own they are unable to satisfy regulatory requirements for the development of MDS (Diebold 2013; Myklebust et al. 2016; McHugh et al. 2013). To meet regulatory requirements, MDS is typically developed in accordance with the V-Model (Kosti 2017; McHugh et al. 2012). However, the V-Model is based upon Royce's Waterfall model which is very rigid in terms of requirements (Özcan-Top and McCaffery 2018a; Zamith and Gonçalves 2018).

To accrue the benefits of ASD, MDS organisations have integrated certain agile practices into the V-Model to achieve specific purposes such as accommodating changing requirements (McHugh et al. 2013) and increasing speed of delivery (Rasmussen et al. 2009). Integrating agile practices into plan-driven methods in MDS development is a common approach to reap the benefits of ASD. However, there is a scarcity of research that specifically discusses using agile practices to address defects in MDS development and details of how these practices were applied for this purpose. This literature review found only one study that used onsite customer or active user involvement to address defects in MDS development (Tang et al. 2018). This study outlined how onsite customer involvement through user design, co-design and participatory design significantly reduced defects.

The limited information on which agile practices can address defects in MDS development and how they can be implemented may contribute to their underutilisation for this purpose. Identifying agile practices that address defects and providing MDS developers with clear implementation guidelines can benefit MDS organisations. MDS developers can select from the identified agile practices and apply them appropriately to address defects during the development process.

Initially, it was thought that agile development could not be applied to the safety-critical MDS domain because of insufficient up front planning (McHugh et al. 2012). The use of agile development was associated with several barriers as discussed by McHugh et al. (2012) and challenges, especially when developing SiMD as discussed by McCaffery et al. (2016).

However, as mentioned earlier, MDS manufacturers can still reap the benefits of ASD by integrating agile practices in their development processes.

2.8.1 Agile Practices Adopted in MDS Development

The need to accrue the benefits of ASD while meeting the regulatory requirements for MDS development has led to the development of tailored approaches for MDS development. Several researchers have proposed different approaches to meet this need. This section provides a review of studies on the adoption of agile practices in the MDS development domain. It examines which agile practices were adopted and whether any were specifically used for addressing defects. Table 2.8 presents a summary of this section.

As mentioned earlier, the literature suggests a growing adoption of agile practices in MDS development for specific project purposes. Out of the 117 papers reviewed in the existing literature, 53 were found to discuss the use of agile practices in MDS development. Of the 53 papers, 34 were relevant to this study as they clearly indicated the agile practices used. The details from the reviewed articles in MDS domain are presented next.

Rasmussen et al. (2009) developed Agile+ by integrating five agile practices into the plan-driven approach at Abbot. Practices included fixed and short duration iterations, automated continuous build, unit test and integration, daily team meetings, and retrospective assessments. Using Agile+ saved 35-50% in cost, accommodated requirements changes, and reduced defects through early working software. The flexibility of ASD to accommodate requirements changes is vital to satisfy customers.

Kane et al. (2006) adapted Scrum and XP practices in biomedical software development projects across six organisations. Practices included iterative development, feature backlog, user stories, refactoring, CI, automated unit tests, open workspace, nightly build and tests, collective code ownership, scrum meeting, developer task selection, mid-range planning, use cases, onsite customer, and automated acceptance testing. PP was used occasionally for training new members or tackling challenging tasks. The team compared progress to estimates at the end of each iteration to provide early indication of completion rate. One project initially used index cards to track requirements but failed due to the customer moving elsewhere. This underscores the importance of customer involvement in software development projects for smooth progress. Kane et al. (2006) concluded that ASD is a valuable approach for biomedical

software projects. They emphasised that organisations wishing to adopt agile practices should prioritise core practices.

Scrum practices were integrated into the document-driven approach in a Danish Pharmaceutical company that had to comply with the FDA's standards for software development (Heeager and Nielsen 2009). These practices included iterative development, software backlog, sprint backlog, burn-down chart, automatic unit testing, timeboxing, stand-up meetings, CI, sub teams/collaboration, and extended stand-up meeting. The development team faced challenges with timeboxing of iterations due to errors discovered during integration. These required prompt fixing, hence using up time allocated for other iterations. This is the consequence of defects slipping through earlier stages of development. Heeager and Nielsen (2009) concluded that while agile and document-driven approaches can coexist, integrating them is challenging. Agile methods are constrained by the need to adhere to FDA standards, bureaucratic project management, and sequential workflows of other groups. These factors disrupt iterations and necessitate frequent hardware and user interface changes. Nevertheless, the iterative development enabled early software development and its evaluation. Self-organising teams enhanced strong ownership and commitment among developers. Task boards and burn down charts enabled visualisation of tasks and progress, while stand-up meetings ensured transparency and timely problem resolution.

To improve the software release cycle time, features delivery, and reduce field defects, the Software Craftsmanship and Scaled Agile methodology were adopted in a software project (Badanahatti and Pillutla 2020). Agile practices from the Scaled Agile Framework® (SAFe®) were used including shorter iterations, program increment planning, cadence-based event, program backlog, team backlog, incremental development, definition of done, user stories, sprint, daily builds, continuous integration/delivery, refactoring, PP, and TDD. SAFe® was chosen for its emphasis on human behaviours and habits. The projects realised a 53% reduction in feature cycle time across releases, improved automated regression failures by 2-10 times for one business, and achieved a 78% reduction in field defect density across businesses.

Selected agile practices from Scrum and XP were integrated with the V-Model, including onsite customer, use cases, user stories, allocate timeboxing, sprint backlog, task board, burn-down chart, daily stand-up meeting, sprint retrospective, test first development, PP, collocated teams, and self-organizing teams (McHugh et al. 2012). This led to the development of the Agile V-Model to accrue the benefits such as increased speed of development, reduction of

costs, and improved efficiency while producing the necessary output for regulatory approval (McHugh et al. 2013). The AV-Model was successfully implemented in an MDS company and helped to streamline requirements changes, reduce rework, and deliver software in a timelier manner (McHugh et al. 2014).

Gary et al. (2011) report on the adoption of agile practices in the Image-Guided Surgical Tool Kit (IGSTK) software project. The team integrated practices from Scrum and XP including product backlog, constant communication, iterative releases, manage source code carefully/coding standard, continuous development, CI, automated testing, sprint backlog, simple design, IGSTK team/Scrum team. The authors challenged the assumption in the software community that document-driven approaches are the only option for safety-critical software development. Their successful integration of agile practices in the IGSTK project demonstrated that agile methods can be used for safety-critical software development.

Cordeiro et al. (2007) developed an agile development approach for embedded systems which was experimented in a case study involving a pulse oximeter. Agile practices utilised included sprint, sprint planning, use cases, sprint review, daily scrum, refactoring, CI, unit testing, coding standards. Unit testing contributed to the correctness of the system. The authors concluded that their methodology reduced design time and development costs while enhancing software modularity and safety. Scrum and XP practices were adopted into the plan driven development process of Class III medical device systems at Medtronic Inc to satisfy customers' demands of innovative and creative solutions (Spence 2005). The practices included planning game, user stories, daily scrum, incremental development, burn-down chart, periodic reflection, TDD, simple design, refactoring, PP, collocated teams, CI, collective code ownership, process ownership, customer role, process discipline, small teams, and coach role. Overall, agile adoption in the organisation led to improvement in velocity, quality and fewer acceptance test defects.

Rottier and Rodrigues (2008) report on the introduction of Scrum and XP agile practices in the development of MDS at Cochlear™. The adopted practices included use cases, user story, sprint, product backlog, sprint backlog, automated testing, TDD, iterative development, refactoring, burn-down chart, short iteration, and acceptance testing. TDD in combination with iterative development improved code quality. The authors concluded that adopting agile practices improved software quality while ensuring the inclusion of relevant functionalities for their customer. Heeager and Nielsen (2020) examined the integration of Scrum practices into

a plan driven model at a pharmaceutical company developing a safety critical clinical device with embedded software. The company wanted to address the complexity and uncertainty in the tasks and requirements. Despite the challenges with documentation, requirements management, testing, and the inexperience of the developers with Scrum, the company achieved agility. However, the company faced difficulties with CI due to numerous software bugs. The practices utilised included iterative development, automated unit testing, incremental development/delivery, planning meeting, sprint backlog, software backlog, task board, burn-down chart, daily stand-up meetings, and CI. Agile practices were used in the development process of an intelligent monitoring system to support semi-autonomous care for persons with intellectual disability (Baljé et al. 2015). Agile practices were adopted to accommodate changing requirements and flexibility in gathering the requirements due to the novelty of the project to stakeholders. Practices included product backlog, sprints, sprint demo, sprint planning, daily stand up, scrum board, burn down chart, and incremental development. The authors concluded that ASD, with its emphasis on user involvement and product visibility, can serve as a change management strategy in healthcare innovation projects.

Guo and Hirschmann (2012) proposed a Safety-oriented Scrum process framework for developing safety-critical systems. The Scrum practices utilised included product backlog, sprint, sprint planning meeting, sprint backlog, sprint review meeting, sprint retrospective meeting, daily Scrum meeting and done criteria. However, as the approach has not yet been implemented, no empirical results have been reported. Manjunath et al. (2013) report on the use of Lean and Agile principles in the development of a medical imaging application. The adopted practices included user stories, task board, simple design, unit testing, scrum meetings, stand-up meetings/scrum of scrums, retrospective meetings, iteration/sprint planning, product backlog, code review, and coding standards. The authors concluded that the adoption of Lean principles enabled the successful completion of the V-Model in each iteration for a set of requirements without schedule overruns.

Scrum practices were integrated into a document-driven method for the development of a clinical product with embedded safety critical software that needed to comply with various quality standards such as those from FDA (Heeager 2014). However, the project faced problems with iterative management of requirements due to the sequential development strategy of the overall project. Practices utilised were iterations, planning meeting, retrospective, product backlog, scrum boards, daily stand-up meeting, integration testing, and

unit testing. Agile practices were adopted in the development of software at Medtronic (Weyrauch 2006). The practices utilised included backlogs, burndown charts, sprint planning, sprints, stories, daily meetings, and pairing. These practices were adopted to demonstrate the immediate benefits particularly because the project was in its early stages of planning and defining with less designing and coding. The authors concluded that the successful adoption of ASD in an organisation depends on effective communication among various stakeholders. In Medtronic's case, this was achieved through use of a framework that outlined principles, practices, and benefits of ASD. Zamith and Gonçalves (2018) present a tailored agile approach based on the Scrum model designed for developing international standards compliant MDS. The practices utilised include product backlog, sprint backlog, definition of done, sprint, sprint planning, sprint review, sprint retrospective, daily Scrum and integration and testing. The authors concluded that the barriers associated with implementing ASD in MDS development, as identified by McHugh et al. (2012), can be addressed through tailoring.

An enhanced Scrum software development life cycle process called R-Scrum was developed at QUMAS (Fitzgerald et al. 2013). R-Scrum was developed to meet regulatory compliance, reduce time to market and solve the problem of large release overhead. Practices utilised were user stories, daily scrum, continuous development, CI, definition of done, iterative development, daily scrum meetings, sprint review, sprint retrospective, product backlog, sprint planning meeting, sprint, refactoring, code review, coding standards, daily stand up meetings, and sprint backlog. The authors concluded that through R-Scrum the organisation achieved "continuous compliance" and "living traceability". Nielsen and Heeager (2017) report on the use of Scrum practices in a software development project to accommodate changing requirements for a clinical device for dispensing drugs, processing information, and patient interaction. They discuss the use of a framework to facilitate the introduction of agile methods. The practices adopted include backlog, burndown chart, test first strategy, iterative development, automated testing, sprint, and sprint review. The authors conclude that their framework provides a clearer understanding of the implications of introducing agile processes in safety-critical software development. A hybrid methodology integrating Scrum practices with the Waterfall model was proposed for software development in the Digital Subtraction Angiography project. This approach aimed to accrue benefits of ASD while meeting FDA requirements (Lin and Fan 2009). The authors concluded that the proposed hybrid approach was the most appropriate for the project.

McCaffery et al. (2016) introduced agile practices into MDevSPICE®, a tool that integrates regulatory requirements for MDS development. Practices included user stories, iterative testing, TDD, automated tests, acceptance tests, refactoring, configuration management, iterative development cycles, CI, sprint planning meetings and continuous requirement prioritization. The authors believe that by introducing agile practices into MDevSPICE®, the process of developing MDS can be done in a flexible manner. Özcan-Top and McCaffery (2017) mapped Scrum practices to MDevSPICE® processes and base practices to demonstrate their alignment. Practices included sprint planning, daily scrum, sprint review, sprint retrospective, and product backlog grooming. The authors concluded that Scrum is related to five processes of the MDevSPICE®. Özcan-Top and McCaffery (2018) mapped the primary practices and the corollary practices of XP to MDevSPICE® processes and base practices to demonstrate their alignment. Practices included weekly cycle, quarterly cycle, whole team, informative workspace, energized work, sit together, pairing, and personal space, PP, slack, ten-minute build, CI, test first programming, incremental design, story, real customer involvement, incremental deployment, team continuity, shrinking teams, root-cause analysis, collective code ownership, code and tests and single code base. The authors concluded that XP aligns with seven processes of MDevSPICE® and that several of its practices are compatible with the framework.

Özcan-Top and McCaffery (2018b) developed a hybrid assessment approach for MDS companies. Practices utilised were weekly cycle, quarterly cycle, whole team, informative workspace, energized work, sit together, pairing, and personal space, PP, slack, ten-minute build, CI, test first programming, incremental design, and user story. The authors discussed the theoretical and field applications of the approach and presented a mapping between XP practices and MDevSPICE® base and specific practices. Demissie et al. (2017) extended the base practices of MDevSPICE® by incorporating agile practices to flexibly address the challenges such as multi-domain communication and co-design in medical embedded software development. Practices utilised were PP, TDD, ATDD, daily scrum, sprint review, and sprint retrospective.

Kircher and Hofman (2012) report on the integration of Scrum into the Product Line Engineering (PLE) approach and use of feature-oriented practice to overcome the challenges of scoping, process efficiency and organisational decomposition which affected the software development at Siemens Healthcare. Practices utilised were scrum teams, platform backlog,

release backlog, burn down chart, iteration, time-boxed development, continuous delivery, CI, operational quality tests, and smoke testing. The authors concluded that PLE and agile methodologies complement each other when implemented within a feature-oriented environment. Additionally, Lean principles help in streamlining medically regulated development by reducing process and documentation overhead while also minimising the challenges of systematic reuse. Mehrfard and Hamou-Lhadj (2011) evaluated the suitability of XP in meeting FDA requirements for MDS development. They found that XP lacks support for requirements traceability analysis, design traceability analysis, source code documentation evaluation, code traceability analysis, source code interface analysis and test traceability analysis. The examined XP practices included PP, collective code ownership, rapid iteration, user stories, TDD, acceptance testing, refactoring, unit testing, and integration testing. The authors concluded that to meet FDA requirements, XP should be extended to include new roles, practices, and artefacts while maintaining XP's values and balancing its agility.

A hybrid model integrating agile and plan driven approach was used at HealthCo to manage non-functional requirements and meet regulatory compliance (Rahy and Bass 2022). The agile practices included user story, product backlog, sprint backlog, product backlog grooming, kanban board, stand-up meetings, sprint retrospective, sprint review, sprint planning meeting, self-organizing teams, sprint, sprint demo, slack, and physical board. The authors concluded that their approach enhances agile methods by incorporating two specific artefacts tailored to regulated environments for facilitating better management of documentation and safety-critical requirements. These artefacts are Documentation Work Item and Safety Critical Work Item.

The AgileSafe framework was developed and used to support integration of agile practices in the software development process for a continuous glucose monitoring-enabled insulin pump system (Łukasiewicz and Górski 2018). AgileSafe adopts practices from SafeScrum (Myklebust et al. 2016) such as functional product backlog and safety product backlog. The authors concluded that AgileSafe has the potential to enhance safety-critical software development. Demissie et al. (2020) developed the Synch Up Process which integrates agile practices from Scrum and XP to support multi-domain stakeholder communication within embedded safety-critical domains. The practices utilised included daily scrum, sprint retrospective, sprint, user stories, on-site customer, the planning game, PP, CI, refactoring, TDD, and acceptance test-driven development (ATDD). The Synch Up process was validated through expert review and was yet to undergo further review via case study. Cawley et al.

(2010) explored the applicability of Lean/Agile Software Development in the development of MDS. They concluded that the potential of Lean Software Development for safety-critical regulated domains needed further detailed investigation.

Suryantara and Andry (2018) developed a medical record application using XP practices to enhance software quality and user satisfaction. Practices included user stories, simple design, PP, CI, acceptance testing, refactoring, and unit testing. The authors emphasised the active involvement of users throughout the development process for feedback to ensure the system meets their needs. Yarlagadda (2017) presented six steps or practices to follow when implementing DevOps in healthcare including continuous planning, version control, continuous integration, automated build, continuous deployment, and constant feedback and tracking. The author highlights DevOps potential in healthcare domain, particularly for pharmaceutical and biotechnology organisations as it accelerates customer feedback, improves software quality and overall customer satisfaction. Solita, a Finish company, utilised DevOps to develop Oravizio, a MDS product that uses machine learning technologies to support Orthopaedic specialists in evaluating joint replacement surgery risk for patients (Granlund et al. 2021). The authors emphasise the potential of continuous software engineering and DevOps in MDS development.

Tang et al. (2018) utilised user involvement or onsite customer in the development of MDS specifically to address defects. The authors clearly outline how user involvement was used in various development stages and methods. For example, during module design stage, user centred design, user co-design, and user participatory design methods were employed. The authors reported a significant reduction in defects as a result of this user involvement.

Table 2.8**:** Agile Practices adoption in MDS development

Author	**Context / Purpose**	**Agile Practices Used**	**Reported Outcomes / Findings**
Rasmussen et al. (2009)	Integrating agile into plan-driven MDS development	Short iterations, continuous build, unit tests, daily meetings, retrospectives	35–50% cost savings; reduced defects; improved handling of requirement changes
Kane et al. (2006)	Scrum & XP in biomedical software across 6 orgs	Iterative dev, backlog, user stories, refactoring, CI, automated tests, PP, onsite customer	Early progress visibility; highlighted importance of customer involvement
Heeager & Nielsen (2009)	Scrum within FDA-compliant document-driven process	Iterative dev, backlogs, burn-down charts, unit testing, CI, stand-ups	Integration challenges; defects caused iteration overruns; improved transparency
Badanahatti & Pillutla (2020)	SAFe® to improve release cycle & reduce defects	Short iterations, PI planning, CI/CD, refactoring, PP, TDD	53% cycle-time reduction; 78% reduction in field defect density
McHugh et al. (2012–2014)	Agile V-Model for regulatory compliance	User stories, timeboxing, task boards, retrospectives, TDD, PP	Streamlined requirements, reduced rework, faster delivery
Gary et al. (2011)	Agile in Image-Guided Surgical Toolkit	Backlog, iterative releases, coding standards, CI, automated testing	Demonstrated agile suitability for safety-critical systems
Cordeiro et al. (2007)	Agile for embedded pulse oximeter	Sprints, planning, reviews, CI, unit testing	Reduced design time; improved modularity and safety
Spence (2005)	Scrum/XP in Class III MDS	Planning game, user stories, TDD, refactoring, CI, PP	Improved velocity, quality, fewer acceptance test defects
Rottier & Rodrigues (2008)	Scrum/XP at Cochlear™	User stories, sprints, TDD, refactoring, acceptance testing	Improved code quality and customer-relevant functionality
Heeager & Nielsen (2020)	Scrum in plan-driven pharma device	Iterative dev, automated tests, CI, backlogs, stand-ups	Achieved agility despite CI challenges due to bugs
Baljé et al. (2015)	Agile for intelligent monitoring system	Backlogs, sprints, demos, stand-ups, burn-down charts	Supported change management; improved user involvement

Guo & Hirschmann (2012)	Safety-oriented Scrum framework	Backlogs, sprint events, done criteria	Conceptual only; no empirical results
Manjunath et al. (2013)	Lean + Agile for imaging application	User stories, task board, unit testing, retrospectives	Enabled V-Model completion without overruns
Heeager (2014)	Scrum in FDA-regulated clinical product	Iterations, planning, retrospectives, integration testing	Requirements management challenges due to sequential constraints
Weyrauch (2006)	Agile at Medtronic	Backlogs, sprints, pairing, daily meetings	Early benefits; communication critical for adoption
Zamith & Gonçalves (2018)	Tailored Scrum for standards-compliant MDS	Backlogs, DoD, sprints, reviews, integration testing	Tailoring helps overcome ASD barriers
Fitzgerald et al. (2013)	R-Scrum for compliance & release overhead	User stories, CI, iterative dev, code review	Achieved continuous compliance and traceability
Nielsen & Heeager (2017)	Scrum for clinical drug-dispensing device	Backlog, burn-down, test-first, automated testing	Framework clarified agile implications in safety-critical work
Lin & Fan (2009)	Hybrid Scrum–Waterfall for angiography system	Scrum practices integrated with Waterfall	Hybrid approach most suitable for FDA compliance
McCaffery et al. (2016)	Agile in MDevSPICE®	User stories, TDD, CI, acceptance tests	Increased flexibility in regulated development
Özcan-Top & McCaffery (2017)	Mapping Scrum to MDevSPICE®	Sprint planning, daily scrum, grooming	Scrum aligns with 5 MDevSPICE® processes
Özcan-Top & McCaffery (2018)	Mapping XP to MDevSPICE®	Pairing, CI, TDD, incremental design, root-cause analysis	XP aligns with 7 processes; many compatible practices
Özcan-Top & McCaffery (2018b)	Hybrid XP-based assessment	Weekly/quarterly cycles, pairing, CI, TDD	Demonstrated theoretical and field applicability
Demissie et al. (2017)	Extending MDevSPICE® with agile	PP, TDD, ATDD, daily scrum	Improved multi-domain communication
Kircher & Hofman (2012)	Scrum + Product Line Engineering	Backlogs, iterations, CI, smoke testing	Agile + PLE improves efficiency and reuse

Mehrfard & Hamou-Lhadj (2011)	XP suitability for FDA	PP, TDD, acceptance testing	XP lacks traceability; needs extensions
Rahy & Bass (2022)	Hybrid agile for non-functional requirements	Backlogs, grooming, Kanban, retrospectives	Added artefacts improved documentation & safety management
Łukasiewicz & Górski (2018)	AgileSafe for insulin pump system	Functional & safety backlogs	Potential to enhance safety-critical development
Demissie et al. (2020)	Synch Up Process for multi-domain work	Daily scrum, PP, CI, TDD, ATDD	Validated via expert review
Cawley et al. (2010)	Lean/Agile applicability in MDS	Lean/Agile principles	Need for deeper investigation
Suryantara & Andry (2018)	XP for medical record system	User stories, PP, CI, acceptance testing	Improved quality and user satisfaction
Yarlagadda (2017)	DevOps in healthcare	CI, automated build, deployment, feedback	Faster feedback, improved quality
Granlund et al. (2021)	DevOps for ML-based MDS	Continuous engineering practices	Demonstrated DevOps potential
Tang et al. (2018)	User involvement to address defects	User-centred design, co-design, participatory design	Significant defect reduction

2.9 Medical Device Software Regulations and Standards

MDS development organisations must comply with regulatory requirements of the countries where their products will be sold. For example, MDS organisations in the United States of America must adhere to regulations set by the FDA, while in the European Union they must adhere to regulations set by the European Commission. Numerous standards, guidance documents, and technical reports have been developed by international organisations to assist MDS organisations in meeting these regulatory requirements. This section provides an overview of certain regulations and standards in the EU and the US.

2.9.1 Regulations

In the EU, the Medical Device Regulation (MDR), commonly cited as Regulation (EU) 2017/745 (EU 2023) is the overarching piece of legislation for MDS organisations. It replaced the previous Medical Device Directives such as EC-MDD-93/42/EEC (European Parliament, Council of the European Union 2007) and Active Implantable Medical Device Directive (AIMDD 90/385/EE) (French-Mowat and Burnett 2012). Although the MDR entered into force on May 25, 2017, it became fully applicable on May 26, 2021, following a transition period and a deferral prompted by the COVID-19 pandemic. The MDR mandates the safety, performance, and transparency of medical devices marketed in the EU, while fostering innovation and improving patient outcomes. Key obligations within the MDR include stricter controls on device traceability, clinical evaluation, and post-market surveillance. Under the Regulation, organisations are required to continuously monitor and report device performance and incidents. Additionally, they are mandated to maintain technical documentation, perform clinical evaluations, and entrust a qualified individual with the responsibility of regulatory compliance. To enhance transparency and traceability, the Regulation introduces a unique device identification system and mandates the use of a central EU Database on Medical Devices (EUDAMED) for public access to safety and performance data. The Regulation further formalises the role of notified bodies which are tasked to assess device conformity before they are marketed.

There are four classifications for medical devices under the MDR. The classifications are based on the device's intended use, the risk it poses to the patient, duration of contact with body, and invasiveness (EU 2023). Medical device classifications and examples are provided in table 2.8.

Table 2.9**:** MDR medical device classification

Class	**Risk level**	**Intended use**	**Example**
Class I	Low	Non-invasive and short-term use	Tongue depressors, surgical instruments
Class IIa	Medium	Invasive for short-term use or active devices for monitoring	Surgical clamp
Class IIb	Higher	Long-term invasive use or active devices for therapy	Ventilators, bone fixation plate
Class III	Highest	Life supporting, life sustaining or implanted long-term use	Heart valves, implantable pacemakers

The MDR, is complemented by the In Vitro Diagnostic Medical Devices Regulation (IVDR), Regulation (EU) 2017/746 (EU 2017), which establishes requirements for in vitro diagnostic devices. It replaced the previous IVDMDD 98/79/EC (European Parliament and the Council of the European Union 1998). Like MDR, IVDR was also enforced in 2017 and became fully applicable on 26 May, 2022 (Kahles et al. 2023). Both the MDR and IVDR stipulate specific obligations for manufacturers, healthcare institutions and other stakeholders involved in the lifecycle of medical devices.

In the US, medical devices are regulated by the FDA through its Center for Devices and Radiological Health (CDRH). There are several regulations that are enforced by the FDA including the Code for Federal Regulations (CFR) 21 Part 820 – Quality System Regulation (FDA 2018), which mandates the safety and effectiveness of devices. In 2024, FDA finalised revisions to the CFR 21 Part 820, introducing the Quality Management System Regulation (QMSR) (FDA 2024b; Holland 2024). The updated regulation aligns the US regulatory requirements with international standards such as the ISO 13485 Medical Devices – Quality Management Systems – Requirements for Regulatory Purposes, and the ISO 14971:2019 Medical Devices – Application of Risk Management (Holland 2024). The QMSR became fully applicable on 2 February 2026 and has replaced the CFR 21 Part 820. Additional FDA regulations include 21 CFR Part 814 – Premarket Approval (PMA) (FDA 2023b; FDA 2002) which mandates manufacturers to submit scientific evidence to prove device safety and effectiveness before marketing. Related to this regulation is the 21 CFR Part 807 – Premarket Notification (FDA 2024a; FDA 2015) which requires manufacturers to demonstrate that a new device is substantially equivalent in terms of safety and effectiveness to a legally marketed device.

Similar to the EU, the FDA classifies medical devices based on the intended use and the risk they pose to patients. There are three categories:

- Class I: low risk devices not intended to support or sustain life e.g., thermometer
- Class II: moderate-risk devices that could pose harm to humans e.g., catheters, syringes
- Class III: high-risk devices intended to support or sustain life e.g. pacemakers

2.9.2 Standards

This section provides an overview of certain standards that apply to MDS development. These documents contain details that MDS organisations should comply with to meet regulatory requirements.

2.9.2.1 IEC 62304 - Medical device software — Software life-cycle processes

The IEC 62304 is one of the most important standards for MDS development organisations. It was first published in 2006 and has undergone revisions over the years with the 62304 Edition 2., published in 2015 as the current version. This version will be replaced with an updated Editon 3 in 2026. The IEC 62304 defines a framework of lifecycle processes, including activities and tasks related to development, maintenance, risk management, configuration management, and problem resolution for software (AAMI/IEC 2015). These processes ensure traceability, accountability, and continuous improvement throughout the development lifecycle. A key aspect of IEC 62304 is its emphasis on safety through the identification of hazards, implementation of controls, and verification of safety throughout the MDS lifecycle. However, the standard does not state how these processes should be carried out, allowing flexibility in implementation.

In concurrence with ISO 14971 - Medical devices — Application of risk management to medical devices (ISO 2019), IEC 62304, section 4.3, classifies MDS into three safety classes – Class A, B, and C, based on potential harm to patients and caregivers. Class A MDS poses no risk of harm, Class B carries risk of non-serious injury, and Class C poses risk of death or serious injury. Importantly, IEC 62304 appears to align with international regulations such as the EU MDR and the FDA 21 CFR Part 820 in various areas including the requirement for documented procedures for software development, validation, maintenance, and risk management. This alignment facilitates regulatory compliance and supports market access across jurisdictions for MDS development organisations adhering to the standard.

While IEC 62304 appears to align with a plan driven methodology for MDS development, it does not mandate any specific methodology. This provides MDS development organisations flexibility, allowing them to map IEC 62304 processes, activities and tasks to their preferred methodologies. Although IEC 62304 offers general descriptions of MDS development life cycle processes, it lacks specific methods for defect prevention. Instead, the standard places greater emphasis on defect identification through testing, specifically during software integration and integration testing and software system testing as outlined in Sections 5.6 and 5.7. However, as stated in Chapter 1, an exclusive reliance on defect identification at later stages can lead to costly rework and increase the risk of defects slipping to the final product. Therefore, approaches that integrate both defect prevention and identification, enabling organisations to address defects throughout the development lifecycle are critical to ensuring the safety and reliability of MDS.

Given the important role of the IEC 62304 in guiding the development of MDS, this research adopts its software development lifecycle processes as the foundation for the AV4AD model. The aim is to assist organisations in proactively addressing defects at all development stages, while also supporting compliance with IEC 62304's documentation and traceability requirements as well as other standards.

2.9.2.2 ISO 14971 - Medical devices — Application of risk management to medical devices

Risk is the product of the likelihood of a harmful incident and its severity (ISO 2019). Risk management is one of the essential activities in MDS development lifecycle. The goal of risk management is to identify, evaluate, scrutinise, estimate, lessen, and control potential problems that may occur in a product (Lindholm et al. 2014). Risk management in the development of MDS is important because of the potential injuries to patients arising due to a software failure. To protect patients from potential catastrophic incidences, regulatory authorities mandate MDS organisations to incorporate risk management in their software development processes. To this end, a standard called the ISO/IEC 14971 - Medical devices — Application of risk management to medical devices (ISO 2019) was produced to assist MDS development organisations comply with regulatory requirements. The ISO/IEC 14971 standard is a de facto standard for risk management in medical devices. It defines risk management terms and provides a framework for the risk management process for identifying risks related to medical devices at all stages of lifecycle of the device. Nonetheless, it does not prescribe implementation details. Research

reveals that MDS development organisations face challenges in identifying an exhaustive set of risks that are relevant to their software product (Lindholm et al. 2014).

2.9.2.3 ISO 62366-1 – Application of Usability Engineering to medical devices

The ISO 62366-1 standard, titled "Application of Usability Engineering to medical devices", defines the usability engineering process or human factors engineering for medical devices (ISO 2015). It provides guidelines for medical device development organisations in analysing, developing, and evaluating the usability of their products, ensuring they achieve adequate usability standard. ISO 62366-1 also specifies methods for these organisations to assess and mitigate risks that might arise with the normal use of their products. The standard promotes the safe and effective interaction between users and medical devices by ensuring use-related risks are minimised. The ISO 62366-1 supports manufacturers in meeting requirements stipulated in the MDR such as those related to risks due to use errors and the emphasis to consider user capabilities and limitations. Similarly, this standard aligns with the 21 CFR 820 especially on usability considerations spelled under design input and validation.

2.9.2.4 AAMI SW91 – Classification of Defects in Health Software

AAMI developed the standard known as AAMI SW91-2018 – Classification of Defects in Health Software in 2018 (ANSI/AAMI 2018), providing a common language for the classification of software defects occurring in health software. SW91 allows developers to perform a root cause analysis using keywords to determine the defect type that they may have experienced in their product. While defect taxonomies such as SW91 help manufacturers to know defect types, actual detection is still reliant on software testing, typically conducted after significant development work has been done.

2.9.2.5 ISO 13485 – Quality Management Systems – Requirements for Regulatory Purposes

The ISO 13485 standard, titled "Quality Management Systems – Requirements for Regulatory Purposes", defines the criteria for a quality management system specific to medical device manufacturers (ISO 2016). This standard outlines the essential requirements for a quality management system, enabling medical device organisations to demonstrate their capability to produce safe and effective medical devices that meet the needs of users and adhere to all relevant regulations. The ISO 13485 closely aligns with the US regulations such as the 21 CFR 820 in areas such as design controls and software validation. Similarly, this standard aligns

closely with the MDR in several areas including the obligation for post-market surveillance through its requirements for feedback and corrective actions.

2.9.2.6 AMMI TIR45

In addition to various international standards related to MDS, several technical reports exist including IEC/TR 80002-3:2014 Medical device software Part 3: Process reference model of medical device software life cycle processes (IEC 62304) (AAMI/IEC 2016) and TIR45. Among the various technical reports, the TIR45 – Guidance on the use of agile practices in the development of medical device software, is particularly significant for MDS organisations adopting agile. The TIR45 focuses on ensuring compliance with international standards when using agile practices. First published in 2012 (AAMI 2012) and updated in 2023 (AAMI 2023), the TIR45 was developed to address the perception that agile methodologies were not suitable for MDS development. The TIR45 provides guidelines on how MDS organisations can benefit from agile practices suitable for MDS development.

Although the TIR45 indicates certain practices that can be used at various stages in relation to the IEC 62304, it does not provide an exhaustive list of all agile practices that could be beneficial for MDS development. As highlighted in section 2.5.2, numerous agile methods exist, each with various practices that could be useful for MDS development. Moreover, although the TIR45 mentions product quality metrics such as “correctness of the product, meeting its specifications, free of defects, reliable”, it does not specifically link any agile practice to these metrics, instead provides general descriptions of the practices in relation to quality. While the TIR45 provides high-level guidance on using agile practices, it would be unreasonable to regard it as a comprehensive development approach. Research indicates that defects that arise during use are often left undetected during the development lifecycle of the software (Gupta, and Sahu 2024). Therefore, research into the use of agile practices to address defects and how to apply them throughout the MDS development lifecycle is essential.

2.10 Identification of suitable Agile Practices

This section outlines the process employed to gather agile practices that can address defects. An initial review was performed to identify agile methodologies. A report by Digital.ai Agility, (formerly VersionOne) indicates several commonly used agile methodologies (Digital.ai Agility 2022). This report served as a starting point to identify other studies discussing these and additional agile methodologies and their agile practices. A total of 34 studies discussing

agile methodologies, and their agile practices were identified and thoroughly reviewed. Examples include Abrahamsson et al. (2002) who provide an extensive review of several agile methodologies that are outlined in the report by Digital.ai and discuss their practices. Azevedo Santos et al. (2013) also discuss various agile methodologies and practices they use. Other studies such as Beck (2000), Haynes and Friedenberg (2006), Jain and Suman (2016), Conboy and Fitzgerald (2010), Bottacci et al. (2022), and Senapathi et al. (2019) were also reviewed. These studies formed the foundation for the initial list of 84 agile practices shown in table 2.10. The study collected practices originally included in different agile methodologies because these practices can be adapted by any manufacturer in their projects. For example, Neumann (2021) adapted TDD, acceptance testing, automated testing, or unit testing into one agile practice called Agile Testing. Myklebust et al. (2018) adapted user story into hazard story. Diebold and Dahlem (2014) also provide a list of practices adapted from the ones originally included in different agile methodologies. A detailed review of various agile methodologies and their practices is provided in section 2.7.2. In addition to the reviewed studies, the TIR45:2023 was reviewed as it provides recommendations for complying with international standards when using agile practices to develop MDS.

Table 2.10**:** Agile methodologies and agile practices

Agile Method	**Agile practices**
XP	planning game, small/short releases, metaphor, simple design, test-driven development, pair programming, refactoring, unit testing, collective ownership, continuous integration, onsite customer, coding standards, 40-hour week, open workspace, user acceptance testing, just rules, use case
Scrum	product backlog, effort estimation, sprint, sprint planning meeting, sprint backlog, daily scrum meeting, sprint review meeting, retrospective, user story
Crystal	staging, code review, monitoring, parallelism and flux, holistic diversity strategy, methodology-tuning technique, user viewings, reflection workshops
Feature Driven Development (FDD)	inspection, feature teams, domain object modelling, developing by feature, individual class ownership, regular builds, configuration management, progress reporting

Rational Unified Process (RUP)	iterative development, manage requirements, use component-based architectures, visually model software, verify software quality, control changes to software
Dynamic System Development Method (DSDM)	active user involvement, team empowerment, frequent delivery/rapid releases, fitness for business, iterative and incremental development, integration testing, stakeholder collaboration, purpose is essential criterion for acceptance of deliverables, requirements are baselined at a high level, all changes during development are reversible
Adaptive SW Development (ASD)	iterative development, feature-based planning, customer focus group reviews
DevOps	continuous planning, continuous development, continuous integration, continuous deployment, continuous testing, continuous delivery, continuous monitoring, continuous feedback
Agile Model Driven Development (AMDD)	active stakeholder participation, apply the right artifact(s), collective ownership, single source information, create several models in parallel, create simple content, depict models simply, display models publicly, iterate to another artifact, model in small increments, model storming, prove it with code, use the simplest tools, test-driven development
Other	Behaviour-Driven Development

The 84 agile practices were examined based on their functional descriptions in the reviewed studies. For practices with different names but with similar functional descriptions, one was retained, and the other was excluded. The practices were also included or excluded based on their functional descriptions in relation to the impact they could have on the requirements, modeling, coding, testing and release stages of the software. Defects most often arise during these stages (Kannan et al. 2019). Finally, 25 agile practices were identified and included in this research. The 25 agile practices are shown in column 2 of table 2.11. Column 3 of table 2.11 shows practices that were excluded based on similar functionality or naming. Where more than one methodology is listed in the first column, it indicates that the same or similar practice was found in those methodologies. For example, collective ownership was found in XP and AMDD. Onsite customer from XP was found with similar names as Active User Involvement, Active Stakeholder participation, and User Viewings in DSDM, AMDD, and Crystal, respectively. Similarly, Small Releases from XP was found with similar names as model in

small increments, frequent delivery, and regular builds in AMDD, DSDM, and FDD, respectively. Practices that were excluded not for addressing defects are shown in table 2.12.

Table 2.11: Agile methodologies, identified practices and excluded similar practices

Methodology	**Included Practice**	**Excluded practices based on similar functionality to the included practices**
XP, DevOps, ASD	Planning Game	continuous planning, feature-based planning
XP, AMDD, DSDM, FDD	Small Release	model in small increments, frequent delivery, regular builds
XP, AMDD	Simple Design	depict models simply
XP, AMDD	TDD	prove it with code
XP	Pair Programming	
XP, AMDD	Refactoring	
XP, DevOps, RUP	Unit Testing	continuous testing, verify software quality
XP, AMDD, FDD	Code ownership	individual class ownership, collective ownership
XP, DevOps	Continuous Integration	continuous integration, continuous testing
XP, DSDM, AMDD, Crystal, DevOps	Onsite Customer/Proxy	active user involvement, active stakeholder participation, user viewings, continuous feedback
XP	Coding Standards	
XP, DSDM	Integration Testing	integration testing
XP, DSDM, Crystal	User Acceptance Testing	purpose is essential criterion for acceptance of deliverables, user viewings, fitness for business
XP	Use Case	manage requirements
Scrum	Product Backlog	
Scrum, DSDM, RUP, AMDD, ASD, DevOps	Sprint/Iteration	iterative development, iterative and incremental development, Continuous development, iterate to another artifact
Scrum, ASD, DevOps	Sprint Planning Meeting	Staging, feature-based planning
Scrum	Sprint Backlog	
Scrum	Daily Stand-up Meeting	
Scrum, Crystal, ASD	Sprint Review Meeting	reflection workshops, focus group reviews
Scrum, Crystal	Retrospective	reflection workshops
Scrum	User Story	
Crystal, FDD	Code Review	inspection
Other, DSDM	BDD	stakeholder collaboration

AMDD, RUP, FDD	Model Storming	create several models in parallel, visually model software, domain object modelling

Table 2.12: Agile methodologies and practices not for addressing defects

Methodology	**Practices not for addressing defects**
XP	metaphor, 40-hour week, open workspace, just rules
Scrum	effort estimation
Crystal	monitoring, parallelism and flux, holistic diversity strategy, methodology-tuning technique
FDD	feature teams, developing by feature, configuration management, progress reporting
RUP	use component-based architectures, control changes to software
DSDM	requirements are baselined at a high level, all changes during development are reversible
DevOPs	continuous development, continuous monitoring, continuous operation
AMDD	apply the right artifact(s), single source information

2.11 An Evaluation of Identified Agile Practices in relation to Addressing Defects

This section presents a review of the 25 agile practices listed in Table 2.11 column 2. This review aims to validate the inclusion of these agile practices by further evaluating their potential to address software defects.

Pair Programming

Pair Programming (PP) is a core practice of XP, that can be utilised in any software development approach (Williams 2001; Vanhanen and Korpi 2007). PP involves two programmers (driver and navigator) working closely together on one computer, with the driver writing code and the navigator examining it to detect defects (Chong et al. 2005; Bryant et al. 2008; Hulkko and Abrahamsson 2005; Vanhanen and Mäntylä 2013). The navigator also thinks of alternatives, and considers strategic design implications to prevent or remove defects early, reducing costs (Williams et al. 2000; Parsons et al. 2008; Sobral 2020; A Dongo et al. 2016). In the context of MDS development, PP can serve as a form of continuous verification and help

meet design output verification requirements, by providing an additional layer of review and validation (ANSI-AAMI 2018).

Test-Driven Development

TDD is a practice for increasing software quality and reliability (Yogesh and Vimala 2020). Developers write unit test cases that fail before implementation code, ensuring frequent testing and continual improvement (Williams et al. 2003; Nagappan et al. 2008; Bhadauria et al. 2020). A failing test describes the intended functionality to be implemented. When implementation code is written, the test passes, guaranteeing the correct and defect-free implementation of the functionality. TDD reduces ambiguity, validates requirements (Maximilien and Williams 2003), and enables early defect detection (Bhadauria et al. 2020). TDD's emphasis on writing tests and running the tests repeatedly provides high capability to find defects in production code and test code (Scanniello et al. 2016). Papis et al. (2022) recommend TDD for software projects where reducing defects and achieving high code coverage is critical.

Unit Testing

Unit testing is a key practice to building high quality, reliable software in ASD and other approaches (Gren and Antinyan 2017; Winkler et al. 2012). It detects many defects through constant use (Tomayko 2002). Developers execute a set of test cases on each unit of software and observe its behaviour (Delgado and Martinez 2013). Failed unit tests identify defects, while successful unit tests confirm desired functionality (Yogesh and Vimala 2020). Programmers fix identified defects ensuring subsequent test case execution passes.

Refactoring

Refactoring is a practice in ASD that improves code quality by making the code easier to understand, modify and maintain (Bekefi et al. 2019). It involves modifying a software system without altering its external behaviour to improve its internal structure (Vonken and Zaidman 2012) and can be used in other development methods (Rech 2007). Refactoring enhances software quality in terms of maintainability, reduced complexity, reusability, efficiency, and extensibility (Reddy and Rao 2009) and helps developers find and eliminate defects (Moser et al. 2008; Sirshar et al. 2019; Gupta and Rohil 2015).

Code Review

Code review is a practice where team members critique changes to a software system to evaluate its technical content and improve quality (McIntosh et al. 2014; Ebert et al. 2021; Morales et al. 2015). It is an effective way to find and fix defects in proposed code changes before they are incorporated into the codebase (Shimagaki et al. 2016). The process requires developers to apply rigour to detect and locate defects and deviations from expected quality (Han et al. 2022). Code reviews are a standard part of modern software engineering workflow (Czerwonka et al. 2015).

Continuous Integration

Continuous Integration (CI) involves running unit and integration tests against new changes to discover software defects earlier and faster (Felidré et al. 2019). Small code components are added immediately once implemented, providing feedback on every change (Vasilescu et al. 2014; Mellegard et al. 2020). Regular integration of small parts allows for quick detection and location of errors (Keskin Kaynak et al. 2019). CI helps developers discover and eliminate defects early, increasing the chance of producing defect-free code (Hilton et al. 2017).

Onsite Customer/User Involvement

Customer involvement is essential in software development to ensure software efficiently performs its intended tasks. Customers can be involved at different stages and to varying degree through strategies such as user-centred design, user co-design, and participatory design (Tang et al. 2018). The practice of onsite customer in ASD allows the customer to steer the project by providing requirements and ensuring quality through acceptance testing (Martin et al. 2004). This reduces uncertainty and identifies rework early (Glaiel et al. 2014). Lack of customer involvement can result in issues with requirements, loss of productivity, and loss of business (Hoda et al. 2010). Customer involvement and clearly understanding customer requirements are vital for building the correct software product (Yaman et al. 2016). Onsite customer with appropriate communication channels is a key element in ASD to ensure the customer provides feedback and proposes necessary changes as development progresses (Hoda et al. 2010; Korkala et al. 2006; Martin et al. 2004).

Integration Testing

Integration Testing aggregates individually developed software components to find defects related to interface, design decisions, and non-functional requirements such as throughput, timing, exception handling etc. (Holling et al. 2016). According to TanLi et al. (2021), 40% of software defects are found in the integration testing.

Behaviour-Driven Development

Behaviour-Driven Development (BDD) is a blend and enhancement of practices from TDD and Acceptance Test-Driven Development (ATDD) (Ali 2019). BDD helps stakeholders to comprehend the behavioural dimension of the software through clear communication of requirements using example interactions in structured natural language, without revealing code-level tests (Medeiros et al. 2017; Binamungu et al. 2020). BDD minimises misunderstanding between stakeholders in software development by emphasising engagement and collaboration, thereby speeding up development and reducing the costs (Nezhad et al. 2018). BDD helps produce software with fewer defects and improved quality (Irshad et al. 2021).

User Story

Correctly defining and understanding requirements is crucial for software development project success (Scott et al. 2021). Requirement quality defects contribute to software projects failures (Lai 2015). User stories in ASD help to define and understand requirements, enabling developers to build the right product and avoid costly defects (Lucassen, Dalpiaz, van der Werf, et al. 2016). User stories also help prioritise requirements and defect fixes (Rodeghero et al. 2017). A user story describes valuable functionality to a user or purchaser of a system (Cohn 2009) and is the most granular representation of a requirement (Lucassen, Dalpiaz, van der Werf, et al. 2016).

Retrospective

A retrospective is a practice used by ASD teams to reflect on their work and improve (Andriyani et al. 2017). It is a special meeting where the team inspects and adapts their methods and collaboration after completing an increment of work. Retrospective meetings provide opportunities to improve product quality, productivity, and estimation capability (Erdoğan et

al. 2018). A retrospective meeting produces a list of action items that outline identified issues and potential solutions (Matthies 2020).

Code Ownership

Code Ownership refers to the responsibility and knowledge developers have over a software component (Bird et al. 2011; Greiler et al. 2015). Fowler (2006) identified three forms of code ownership: strong, weak, and collective. Strong code ownership assigns individual modules to specific developers, who are the only ones allowed to make changes. Weak code ownership also assigns modules to owners but allows other developers to make changes. Collective code ownership abandons individual ownership, allowing anyone on the team to make changes anywhere (Russo et al. 2010). Collective code ownership is promoted in XP (Livermore 2008; Shukla and Williams 2002), while strong and weak code ownership have been studied in relation to software quality aspects such as defects.

Daily Stand-up Meeting

A daily stand-up meeting is a practice used in ASD to improve communication and coordination among the development team members (Stray et al. 2017; Stray et al. 2012; Wiesche 2021). It is a short, informal meeting where team members discuss their progress and problems preventing them from making progress (Stray et al. 2017). The meeting promotes continuous feedback and refinement in software development projects (Wiesche 2021).

Rapid/Shorter/Small Releases or Frequent Builds

Small releases help developers to reduce the time to receive feedback from the customers, enabling them to quickly learn and react to the customer needs (Khomh et al. 2012). Customers can potentially identify defects and provide rapid corrective feedback to developers (Fan 2018), reducing the time for testing and fixing defects (Mantyla et al. 2013).

Simple Design

The practice of simple design aims to develop the simplest possible solution by removing unnecessary complexity and extra code at an early stage (Abrahamsson et al. 2002). Academic literature on simple design is scarce, but Jonsson et al. (2012) advocate that it can reinforce other practices such as PP, refactoring, coding standards, and collective code ownership.

Coding Standards

Coding standards are collections of rules, guidelines, and best practices that developers follow when writing code (Wang et al. 2008). They ensure that code is readable, maintainable, and extensible while reducing the risk of bugs (Popic et al. 2018; Wang et al. 2008; Pieterse 2008), leading to more reliable code and cost savings in terms of rework (Pruim et al. 2022). Doody (2004) and Jonsson et al. (2012) state that coding standards can reinforce other practices such as PP, collective code ownership, refactoring, small releases (Haynes and Friedenberg 2006), and code review (Ochodek et al. 2020).

Use Case

A use case is a sequence of actions performed by the system which combined produce a result of value to the user (Collard 1999). Use Cases form the basis for function requirement development, system design and testing by specifying system behaviour from a user's perspective (Liu et al. 2014). They define a sequence of interactions occurring between users and systems in a specific setting to achieve a desired goal. Use cases serve as a tool for identifying, specifying, and verifying the requirements of a software system (Barros-Justo et al. 2019).

Product Backlog

The Product Backlog is a prioritized list of requirements for software product development (Vijay and Ganapathy 2014; Kokkola 2004). It contains all features, functions, technologies, and bug fixes to be made in future releases (Kokkola 2004). The product backlog is a tool used in ASD projects to manage and prioritise items that need to be developed for a software system, guaranteeing that the most crucial requirements are tackled first (Perkusich et al. 2017). This helps developers to focus on releasing the most important functionalities to the client.

Sprint Backlog

The Sprint Backlog consists a list of product backlog items selected for development in a single sprint, including a plan for releasing a potentially shippable product increment (Vijay and Ganapathy 2014). Sprint Backlog is used by the development team and depicts the development work planned for the sprint (Koi-Akrofi et al. 2019).

Sprint

The purpose of a sprint is to develop a new piece of working software that adds value for the customer (Meier and Ivarsson 2013). It is a time frame during which the development team must complete specific set of tasks and make them ready for review at the end (Hema et al. 2020). Sprints are never extended regardless of whether the work has not been completed (Deemer et al. 2012).

Sprint Planning Meeting

Sprint Planning Meeting is a practice where the development team and product owner select tasks to deliver during an iteration (Meier and Ivarsson 2013). During the sprint planning meeting, work items such as bug fixes are selected from the product backlog into the sprint backlog. The sprint planning meeting consists of two parts. The first part of the meeting focusses on what items to prioritize for the next sprint, while the second part focuses on planning how to implement the items the team has decided to tackle (Deemer et al. 2012).

Sprint Review Meeting

Sprint Review Meeting is a practice where the development team demonstrates completed work to the product owner and collects feedback about the system under development (Meier and Ivarsson 2013). The feedback is categorised as Corrective, Perfective, and Progressive (Fan 2018). Corrective feedback is about detected defects that should be fixed. Perfective feedback suggests changes for better system performance or user experience. Progressive feedback suggests new features for development.

Model Storming

Model Storming is a core practice of Agile Model-Driven Development (AMDD) where developers and stakeholders meet for a few minutes to analyse requirements, create a model/sketch, and then continue with coding (Ambler 2007). This allows for quick resolution of issues such as requirements defects (Alshazly et al. 2014) before coding.

Planning Game

In Planning Game, user stories are written, cost-estimated, prioritised and selected for the next iteration based on the set priorities (Grisham and Perry 2005). This practice in XP projects

promotes communication among stakeholders (Koskela et al. 2004) and is similar to sprint planning in prioritising items such as bug fixes for the next iteration.

User Acceptance Testing

User Acceptance Testing (UAT) in ASD allows for regularly testing of the software by the customer to confirm it meets expectations (Martin et al. 2004). Therefore, UAT provides assurance that the system has the required features and is fit for purpose (Pandit and Tahiliani 2015; Miller and Collins 2002). The literature reveals that UAT is important in reducing the risk of developing a system that does not meet customer expectations, and that if employed early, it can identify defects and have them fixed early, thereby avoiding costly rework later.

2.12 Summary

The literature review examined the problem of defects in MDS and their consequences for both patients and manufacturers, as well as techniques used to address these defects. It analysed various SDLC models to identify those used in MDS development. The review revealed three common plan-driven approaches, namely Waterfall, V-Model, and Spiral Model. The findings revealed that among the many plan-driven approaches, Waterfall is the oldest, with the V-Model, Spiral Model and others developed to address its limitations.

A common theme in plan-driven approaches is the importance of following a defined process to get the work done. Additionally, detailed documentation and extensive design are considered essential quality aspects within these approaches. However, all plan-driven approaches share a common weakness of being rigid due to their strict adherence to plans, which limits adaptation to changing requirements and opportunities. Leveraging opportunities that emerge in a dynamic business environment can be challenging once software development has commenced. Despite their weaknesses, plan-driven approaches remain dominant in various software development domains, especially safety-critical ones, due to their capability to produce deliverables required for regulatory compliance. Among the various plan-driven approaches, the review revealed that the V-Model is particularly the best choice in domains such as aerospace, automotive, railway, and medical. The V-Model is the most commonly used in MDS development.

The literature review also focused on ASD methodologies and their practices, revealing benefits related to software quality. These methodologies share common values, principles and

characteristics that facilitate faster delivery and shorten project lifecycles. Additionally, agile methodologies offer the flexibility to start development without a complete upfront plan, in contrast to plan-driven approaches. Agile methodologies differ in terms of practices or activities employed to execute a software project. For example, while Scrum utilises backlogs, RUP and AMDD emphasise the creation of models. A common theme across ASD methodologies is the importance of valuing and trusting people to achieve project success.

While agile methodologies are advocated for software quality, the findings revealed that agile practices are commonly integrated with plan-driven approaches in MDS development instead of being used exclusively. Consequently, various hybrid models have been developed that integrate agile practices with plan-driven approaches for purposes such as reducing time to market, lowering development costs, and accommodating changing requirements.

Although agile practices such as TDD, PP, unit testing, and integration testing are mainly advocated to address defects, no study specifically discusses using them for this purpose despite their adoption in MDS development. This could be an oversight, leading one to assume that they were used for this purpose based on their functions. However, the lack of detail on how they were used does not help first-time adopters. It may also be inaccurate to suppose that their adoption addressed defects considering that some practices have several effects. For example, TDD is associated with many other effects such as external quality, code complexity, coupling, size, cohesion, code coverage (Khanam and Ahsan 2017), and productivity (Tosun et al. 2017; Madeyski and Szała 2007). Similarly, PP is associated with other aspects such as communication, knowledge of work, and productivity (Vanhanen and Mäntylä 2013; Layman et al. 2004; Lui and Chan 2006; Balijepally et al. 2009; Stapel et al. 2010).

Quality improvement is mentioned in some of the reviewed studies but not clearly linked to defects. Moreover, any quality improvement is attributed to the general adoption of agile practices utilised. One study reported a 78% reduction in field defects across businesses using the SAFe® framework (Badanahatti and Pillutla 2020), but it is unclear which specific agile practice reduced defects, because not all projects may need to adopt all the agile practices of the SAFe® framework. Similarly, another study mentioned a reduction in defects through early working software enabled by five agile practices adopted in the project (Rasmussen et al. 2009). Since software quality encompasses several metrics such as defect density, code quality, usability, maintainability, and security (Rashid et al. 2019), first-time adopters would benefit from explicit guidance on which agile practices to implement and how to apply them to address

specific quality metrics. Most of the reviewed studies discuss integrating agile practices into the plan driven approaches to balance the need for agility with the need for regulatory compliance. However, only one study specifically discussed using an agile practice to address defects in MDS development. This study used onsite customer or user involvement and reported significant reduction in defects as a result of this practice.

Selecting appropriate agile practice(s) to address defects throughout the MDS development lifecycle is crucial to avert costly consequences. The results of the literature review are important as they serve as the foundation for the proposed solution in this research. These findings provide valuable insights into the development of a suitable solution for addressing defects in MDS development. As the adoption of agile practices in MDS development continues to grow, there is a need to develop optimal approaches that leverage these practices to adequately and proactively address defects.

This chapter also presented an overview of medical device regulations in the EU and the US. In the EU, the MDR is the overarching legislation that governs the marketing of medical devices. In the US, several regulations including the recently introduced QMSR are enforced by the FDA for organisations seeking market access. The chapter also reviewed various international standards including IEC62304 which defines software lifecycle processes for MDS. Furthermore, this chapter identified and evaluated potential agile practices for addressing defects.

3 Research Methodology

3.1 Introduction

This chapter presents the research methodology used to realise the research objectives outlined in Chapter 1 section 1.6.3. Research methodology is a way to systematically solve the research problem (Patel and Patel 2019). It provides the theoretical foundation for understanding which method, set of methods or best practices can be applied to specific cases (Igwenagu 2016). Saunders et al. (2019) depicted the overall research methodology in the form of an "onion", consisting of several layers that must be peeled away to get to the inner part that represents thoughts regarding the research problem. A clear understanding of these layers is key to the choice of the research methodology to address the research questions and attain the research objectives. The following subsections provide an overview of the different parts of the research "onion" by Saunders et al (2019) (see Figure 3.1) and outline the components that form the research methodology for this research.

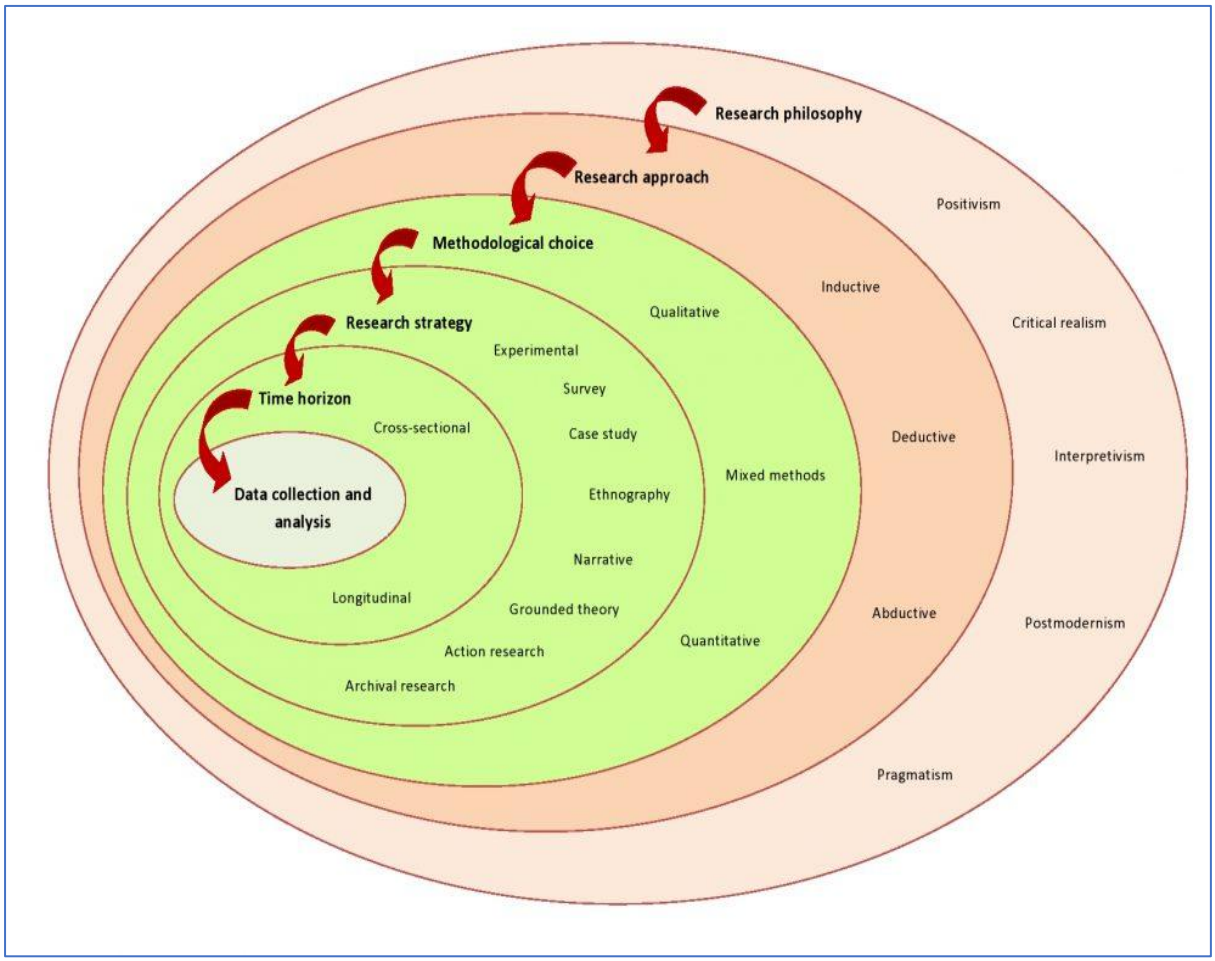


Figure 3.1: The Research 'onion' by Saunders et al. (2019)

3.2 Research Paradigms

Any research is guided by a paradigm or philosophy that determines the researcher's perspective on a particular problem. This paradigm influences methodological decisions in data collection, analysis, interpretation, and dissemination of research findings (Maksimović and Evtimov 2023). The outermost part of the "onion" is the research philosophy. It relates to the development and the nature of knowledge (Saunders et al. 2019). Different philosophical perspectives encompass various ideas about the nature of reality and the methods through which humans acquire knowledge of that reality. Research philosophy is classified into ontology, epistemology, and axiology, each concerned with the researcher's assumptions (Park, Konge and Artino, 2020). Ontology focuses on the nature of reality and raises questions about the researcher's assumptions concerning how the world operates (Ryan 2006; Saunders et al. 2019). Epistemology relates to suitability of the knowledge developed from the research process (Ryan 2006; Saunders et al. 2019). Axiology studies judgements about value and concerns the credibility of research results (Ryan 2006; Saunders et al. 2019). Researchers articulate their values as a basis for making judgements about their research and methodologies (Ryan 2006; Saunders et al. 2019).

The literature identifies several paradigms that scholars may adopt when framing their inquiries, including positivism (Saunders et al. 2019; Park et al. 2020; Ramsberg 2018; Dudovskiy 2022; Ryan 2006; Hiller 2016; Kivunja and Kuyini 2017), post-positivism (Hiller 2016; Panhwar et al. 2017; Maksimović and Evtimov 2023; Hassmén et al. 2016; Writing Commons 2022), realism (Eastwood et al. 2018; Saunders et al. 2019), interpretivism (Žukauskas et al. 2018; Saunders et al. 2019; Brown 2015), constructivism (Kaushik and Walsh 2019; Saunders et al. 2019; Adom et al. 2016; Creswell and Creswell 2019), and pragmatism (Žukauskas et al. 2018; Saunders et al. 2019). A review of these paradigms was performed to determine the most appropriate philosophical foundation for this study. Based on this review, pragmatism was identified as the most suitable paradigm, given its emphasis on practical problem-solving and methodological flexibility. The subsequent subsection provides an overview of pragmatism and outlines the rationale for its selection as the guiding paradigm for this research.

3.2.1 Pragmatism

The pragmatism stance claims that the choice of research philosophy is mostly determined by the research problem and research question (Žukauskas et al. 2018; Saunders et al. 2019). This

paradigm values practical results, considering them as the important criteria for knowledge, meaning, and value. According to Kaushik and Walsh (2019), pragmatism integrates both post-positivist and constructivist perspectives, providing a more flexible and reflexive approach to research design. This allows researchers to implement mixed-methods or multiple methods suitable for a particular research problem (Hassmén et al. 2016; Kaushik and Walsh 2019). Pragmatism has established itself as a dominant paradigm, proving useful in various research fields (Kaushik and Walsh 2019). It focuses on generating practical knowledge that is evaluated in terms of its ability to solve real-word problems (Dube et al. 2024). Within the pragmatism philosophy, the generated knowledge is expected to offer immediate and practical utility, enabling individuals or organisations to navigate and alleviate the challenges they currently face (Dube et al. 2024). This philosophy's orientation towards problem-solving aligns closely with the aims of Design Science Research (DSR), which essentially focuses on creating innovative artefacts and solutions to real-world problems (Hevner et al. 2004). Therefore, pragmatism provides a strong foundation for DSR which underpins the methodological positioning of this research.

A pragmatist perspective permits the use of various data collection and analysis methods to best address the research problem (Creswell and Creswell 2019). The pragmatic philosophy is adopted in this research as it seeks to develop a practical solution to the real-world problem of defects in MDS development. The pragmatism paradigm aligns with the Design Science Research (DSR) strategy taken in this research, particularly its emphasis on creating and evaluating artefacts through iterative cycles of inquiry, allowing the researcher to utilise multiple methods based on their usefulness in addressing the research problem.

Additionally, the pragmatic stance is adopted in this research as it allows for a mixed-methods approach, incorporating both quantitative and qualitative methods. The mixed-methods approach enabled the researcher to gain deeper insights and increase validity as the methods used complemented each other. This study began with an exploratory approach, followed by a survey that combined elements of both quantitative and qualitative methods to explore current practices in using agile practices. Building on these insights, the AV4AD model was subsequently developed, evaluated through qualitative semi-structured interviews, and finally validated through desk-based implementations. These processes reflect a methodological diversity that pragmatism explicitly supports. Therefore, the pragmatist stance is taken because

it provides an appropriate philosophical foundation for this research, supporting the develop of an artefact that is both theoretically grounded and practically relevant to the MDS domain.

3.3 Research Approaches

Dudovskiy (2022) describes a research approach as "a general plan and procedure for conducting the study". The literature identifies three primary research approaches: deductive, inductive, and abductive (Saunders et al. 2019). This research utilises both deductive and inductive approaches to acquire new knowledge. In essence, its overall orientation is abductive, as the abductive approach allows the researcher to combine aspects of both inductive and deductive approaches. The following subsections discuss the two approaches and explain why they are suitable for this research.

3.3.1 Deductive Research Approach

Deductive research approach focuses on developing a hypothesis that is based on an existing theory and then producing a research strategy to test that hypothesis (Dudovskiy 2022). It involves the development of a theory that is subjected to a rigorous test (Saunders et al. 2019). According to Gabriel (2013), deductive research approach generally begins with a hypothesis and emphasises causality. This research adopts the deductive approach as it seeks to investigate whether agile practices can help to address defects in MDS development. The evidence from the non-MDS domain, discovered through the literature review, on using agile practices to address software defects serves as a basis for adopting a deductive approach in this research.

3.3.2 Inductive Research Approach

The inductive research approach begins with an abstract, logical relationship among concepts then progresses towards concrete empirical evidence (Ali and Birley 1999). Researchers start with a set of observations or questions and theories are proposed towards the end of the research process based on these observations (Zalaghi and Khazaei 2016; Saunders et al. 2019). According to Gabriel (2013), an inductive approach typically uses research questions to narrow the study's scope. This research adopts the inductive approach to acquire and generate new knowledge on using agile practices to address defects in MDS development. Initially, an exploratory study was conducted to investigate the impact of MDS on healthcare. The findings of this exploratory study laid the foundation for the research. Subsequently, research questions and objectives were developed to be addressed in this study.

3.4 Research Strategies

A research strategy is a general plan of how the researcher will conduct the study to answer the research questions (Johannesson and Perjons 2014). The literature provides various research strategies that researchers can utilise to generate new knowledge. Saunders et al. (2019) present several research strategies including Experiment, Survey (Glasow 2005; Ponto 2015), Archival and documentary research (Ventresca and Mohr 2017), Case Study (Wedawatta et al. 2011; Crowe et al. 2011; Goundar 2012), Ethnography (Reeves et al. 2013; Johannesson and Perjons 2014), Action Research (Sein et al. 2011; Dudovskiy 2022; Lau 1999; Davison et al. 2004; Iivari and Venable 2009; Johannesson and Perjons 2014), Grounded Theory (Johannesson and Perjons 2014), and Narrative Inquiry. In addition, Hevner et al. (2004) and vom Brocke et al. (2020) present the Design Science Research (DSR) also known as Design Science that focuses on the creation and evaluation of innovative artefacts.

According to Ponto (2015), the choice of a research strategy depends on various factors that include the purpose of the research, the type of research questions to be answered, and the availability of resources. A critical review of the strategies identified in the literature was undertaken to determine the most suitable strategy for this study. Based on this review, DSR was identified as the most appropriate strategy, given the study's focus on developing and evaluating a novel artefact. The subsequent subsection discusses DSR and outlines the specific DSR methodology adopted for this research.

3.4.1 Design Science Research

Design Science Research (DSR) also known as Design Science, is the predominant strategy in the Information Systems (IS) field (Hevner et al. 2004; vom Brocke et al. 2020). DSR is "a problem-solving paradigm that seeks to enhance human knowledge via the creation of innovative artifacts" (vom Brocke et al. 2020). According to Hevner et al. (2004), DSR aims to enhance human knowledge through the creation of innovative artifacts and the generation of design knowledge via innovative solutions to real-world problems. DSR does not assume a specific client but rather aims to address a class of problems in a way that it is useful for specific clients (Iivari and Venable, 2009). This research aligns with the DSR approach, as it does not seek to address MDS defects faced by a specific client but aims to develop a solution applicable to any client seeking to address MDS defects. There are various forms of DSR strategies in the literature, such as Action Design Research (ADR) developed by Sein et al. (2011) and Design Science Research Methodology (DSRM) developed by Peffers et al. (2007). The following

subsections provide an overview of these two strategies and explain why the DSRM is the most suitable strategy for this study.

3.4.1.1 Action Design Research

ADR is a variant of Design Science Research (Haj-Bolouri et al. 2017) that derives its characteristics from Action Research (Sein et al. 2011). It helps stakeholders to shape the research by emphasising the build-intervention-evaluation cycles (Haj-Bolouri et al. 2017). ADR solves real world problems through collaboration between industry stakeholders and researchers, demanding that researchers consider the problem as a representative of a category of problems. Additionally, there should be a balance in terms of expectations from industry stakeholders and researchers. ADR focuses on two main challenges as follows:

1) tackling a problem situation faced in a certain organisational setting through intervention
2) building and evaluating an IT-artifact that solves the category of problems represented by an encountered situation

Sein et al. (2011) proposed and developed ADR when they discovered that Design Research methods failed "to recognise that the artifact emerges from interaction with the organisational context even when its initial design is guided by the researchers' intent". They define ADR as "a research method for generating prescriptive design knowledge through building and evaluating ensemble IT artifacts in an organisational setting". ADR encompasses three interconnected activities of building the artefact, intervening in the organisation, and concurrent evaluation of the artefact. ADR emphasises collaboration between industry and research community in artefact development (Sein et al. 2011). This research does not align with ADR as MDS organisations are only involved in the evaluation and validation of the AV4AD model, not its actual development.

3.4.1.2 Design Science Research Methodology

The DSRM incorporates principles, practices, and procedures to carry out DSR and offers researchers the flexibility to initiate the development process from multiple entry points (Peffers et al. 2007). Additionally, DSRM presents a nominal process for conducting DSR (Peffers et al. 2007). The DSRM process model consists of six consecutive activities as shown in figure 3.2.

The ADR emphasises that the activities of building and evaluation should be intertwined, which contrasts with the structure of this research. In this study, the two activities are sequential: the AV4AD model is developed first, and then evaluation is done after. Moreover, ADR emphasises a specific client and requires that practitioners from the client organisation be involved throughout the project. This research does not assume any specific client and the practitioners are only involved during the evaluation and validation of the model through semi-structured interviews and implementation, respectively. Therefore, ADR is not suitable for this research.

The DSRM is chosen as the overarching strategy for this research because its sequential activities align appropriately with the structure of this study. This research originated from the problems of defects in MDS which were discovered during the initial exploratory study the researcher performed. Based on the information gathered, a decision was made to develop a solution applicable to any MDS development organisation, rather than limiting the intervention to a specific client organisation. The AV4AD model is designed for use by any organisation developing MDS, irrespective of organisational size or MDS classification. It was also determined that once a solution is developed, it should be evaluated by multiple MDS organisations, with the collected feedback informing subsequent refinements. Subsequent validation should then be carried out in multiple MDS organisations to confirm its efficacy.

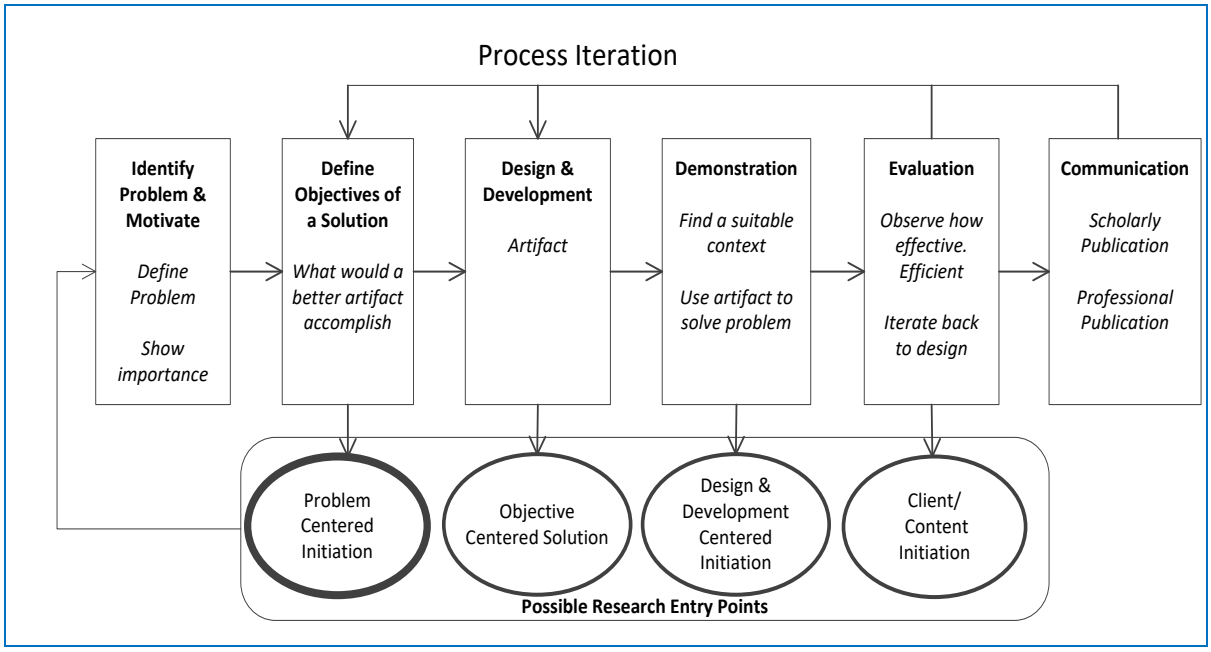


Figure 3.2: The DSRM Process Model adopted from Peffers *et al.* (2007)

3.5 Research Choices

Saunders et al. (2019) provide various methods that a researcher can utilise from both qualitative and quantitative perspectives. In the mono method, the researcher chooses one data collection technique, either qualitative or quantitative. In the mixed method, the researcher uses both qualitative and quantitative methods. In the multi-method, the researcher uses more than one data collection method but aligns them to either qualitative or quantitative. The mixed method is chosen and utilised in this research as it enhances validity by allowing the methods to complement each other.

3.6 Time Horizon

Saunders et al. (2019) define two time horizons for research: cross-sectional and longitudinal. A cross-sectional time horizon involves the study of a particular phenomenon at a specific or fixed time, often using survey, qualitative or mixed methods. It is usually chosen due to time constraints on the researcher. In contrast, a longitudinal time horizon represents a series of snapshots and events over a given period, signifying research conducted over a prolonged period. This study is cross-sectional due to time constraints typical of academic research.

3.7 Application of Design Science Research Methodology

This section details how the DSRM process model is used in this research. This research adopts a problem-centred approach which starts with activity 1 as shown in figure 3.2. The details of how the DSRM is employed to accomplish this research are presented under each activity in the following subsections.

3.7.1 Activity 1: Problem Identification and Motivation

This activity involves defining the specific research problem and justifying the value of a solution. The problem addressed in this research was identified through an initial exploratory literature review which revealed that defects in MDS resulted in patient injuries, deaths, and an increasing trend of medical devices recalls. The review also revealed that defects occurring after release often originate in the early stages of the software development lifecycle. Accordingly, the review aimed to investigate how MDS is developed and identify existing solutions for addressing defects, their limitations, and determine the best approach to addressing defects in MDS development. The AV4AD model is developed in this research to assist MDS organisations in addressing defects by preventing them in the early stages of MDS development lifecycle and identifying any that may occur in later stages.

3.7.2 Activity 2: Define the Objectives for a Solution

This activity aims to infer the objectives of a solution from the defined problem and knowledge of what is possible and feasible. In response to the problem of defects in MDS development, this research aimed to develop the AV4AD model that integrates suitable agile practices to address defects in MDS development lifecycle by preventing them early on and identifying any that may occur in later stages. To achieve this aim, six research objectives were developed and are presented in Chapter 1, section 1.6.3.

3.7.3 Activity 3: Design and Development

This activity is concerned with the design and development of an artefact. It includes determining the artefact's desired functionality and architecture and then creating the actual artefact. According to Peffers et al. (2007) and Hevner et al. (2004), potential DSR artefacts include constructs, models, methods, and instantiations. In this research, the artefact developed is a model- the Agile V-Model for Addressing Defects (AV4AD), designed to prevent and identify defects in MDS development. The AV4AD model incorporates agile practices to

prevent defects in the early stages of MDS development lifecycle and identify defects in later stages.

3.7.4 Activity 4: Demonstration

This activity demonstrates the effectiveness of the artefact in solving one or more instances of the problem it was designed to address. The AV4AD model was implemented within two MDS organisations to assess its ability to address defects. In addition to the researcher's availability during implementation of the model, developers received guidance on using the model for optimal results.

3.7.5 Activity 5: Evaluation

This activity focuses on observing and evaluating how well the artefact supports the solution to the problem. Evaluating the artifact involves comparing its intended objectives with the actual results observed during its use in the demonstration. To evaluate the AV4AD model, semi-structured interviews were conducted with six MDS developers from different MDS organisations. Their feedback was assessed for potential improvements to the model. A detailed process of the semi-structured interviews conducted in this research is presented in section 3.10.

3.7.6 Activity 6: Communication

This activity involves communicating the problem's significance, the artifact's utility and novelty, the rigor of its design, and its effectiveness to researchers and other relevant audiences. The results of this research are communicated through the AV4AD model, which is developed to assist MDS organisations in addressing defects during MDS development. This thesis also serves as a means of communicating the problem that the study intended to address and how the solution was realised. Additionally, three peer-reviewed conference papers have been published to communicate parts of this research.

3.8 Research Reliability and Validity

3.8.1 Reliability

In quantitative research, reliability refers to the consistency, dependability and replicability of the research findings, with a primary focus on achieving identical results (Zohrabi 2013; Olmsted 2024). The numerical nature of the data in quantitative research makes this process

relatively straightforward to accomplish. However, in qualitative research, achieving identical results can be challenging due to the narrative nature of the data. To address this challenge, researchers emphasise that dependability and consistency are the most appropriate for achieving reliability in qualitive research (Zohrabi 2013). Consequently, given the qualitative nature of the data collected in this research, it is reasonable to consider reliability based on these two aspects.

To achieve reliability in this research, the developed AV4AD model is founded on the V-Model, which is widely used in the development of MDS and proven for its capability to produce deliverables for regulatory compliance. Additionally, the AV4AD model is designed to align with the IEC62034 as it adopts the lifecycle processes of this standard. Therefore, the model assists MDS organisations to address defects while also ensuring compliance with the IEC62304 requirements. Moreover, standards relevant to MDS development process have been incorporated into the AV4AD model for reference purposes. The model underwent evaluation and validation by experienced MDS developers who confirmed its importance and effectiveness in addressing defects. Additionally, the data collection instruments for this research underwent reviews by five researchers from the Regulated Software Research Centre at Dundalk Institute of Technology (DkIT). These reviews provided feedback that led to improvements in the instruments in terms of clarity, consistency, and flow of questions. Furthermore, transcriptions from the semi-structured interviews were documented verbatim and shared with interviewees for verification.

3.8.2 Validity

According to Golafshani (2015) “Validity determines whether the research truly measures that which it was intended to measure or how truthful the research results are”. In this research, validity is achieved through the following criteria: *content validity*, *internal validity*, *external validity*, and *utility*. These criteria are discussed in the following subsections.

3.8.2.1 Content Validity

Content validity is defined as “the degree to which elements of an assessment instrument are relevant to and representative of the targeted construct for a particular assessment purpose” (Yusoff 2019). In this research, content validity was achieved through the development and review of data collection instruments, namely survey questionnaire and interview guide. These instruments underwent evaluations by five researchers from the Regulated Software Research

Centre at DkIT. Feedback from these researchers led to improvements in the questions and their flow. Additionally, the instruments were piloted before the actual implementation. The pilot survey and interviews facilitated further refinements of the instruments, ensuring the collection of rich data.

3.8.2.2 Internal Validity

Internal validity refers to "the degree to which the researcher observes and measures what is supposed to be measured" (Zohrabi 2013). In this research, internal validity was achieved through three methods, namely triangulation, member checking, and participatory or collaborative mode of research (Zohrabi 2013). Triangulation involves the use of multiple methods or data sources in research (Carter et al. 2014). In this research, two types of triangulations were used: data triangulation and method triangulation. Data triangulation, which emphasises using multiple sources of data to investigate a phenomenon (Donkoh 2023), was achieved through the use of the survey, semi-structured interviews, implementation of the AV4AD model, as well as the literature review. For instance, both the literature review and survey findings aligned in highlighting the common use of plan driven approaches in MDS development. The survey further revealed exclusive use of agile methods within certain organisations. This triangulation informed the development of a suitable solution adaptable to both plan-driven and agile environments.

Method triangulation was realised through the pragmatic stance taken in this research which facilitated the use of a mix of elements from both quantitative and qualitative research approaches. Member checking involves taking data and interpretations back to participants for confirmation and validation (Zohrabi 2013). In this research, the transcriptions from the semi-structured interviews were shared with the interviewees for their review and confirmation. Additionally, as the AV4AD model was partly developed based on the data gathered through the survey of MDS organisations, their involvement in its evaluation and validation served as part of the member checking by confirming the efficacy of the model in addressing defects in their organisations. Participatory or collaborative research is about involving participants throughout the research. This was partly achieved in this research through the involvement of MDS organisations in the survey, semi-structured interviews, and the implementation of the AV4AD model.

3.8.2.3 External Validity

External validity is concerned with the applicability of the research findings in other settings or a wider domain (Zohrabi 2013). To address this concern, the AV4AD model developed in this research is founded on the V-Model, which is widely used and proven for its capability to produce deliverables for regulatory compliance in both MDS and other safety-critical software domains. The AV4AD model underwent evaluation through semi-structured interviews with six developers from different MDS organisations in Ireland. The model was further validated through implementation in two MDS organisations. While the focus of this research was on addressing defects in MDS development, the findings have broader applicability. Organisations in other safety-critical software development domains, facing similar challenges related to defects, may also benefit from adopting the AV4AD model.

Furthermore, the findings of this research are applicable to the non-critical software domain. Defects in non-critical software systems can also lead to project failure and financial loss. By adopting the AV4AD model, organisations in this domain can effectively address defects and avoid such consequences. Therefore, the AV4AD model holds potential relevance for addressing defects across a range of software development environments.

That said, a potential limitation affecting the generalisability of the findings of this research still exists. Although the model has been evaluated through semi-structured interviews, validated in two SMEs, and published in international conference proceedings, broader implementation is required. Further implementation in a larger number of organisations of varying sizes in different geographical locations would strengthen claims of generalisability.

3.8.2.4 Utility

Utility describes the degree of usefulness the evaluation findings have for a broader context (Zohrabi 2013). The AV4AD model was validated through implementation in two SMEs, both of which developed Class I category MDS. Validation findings in both organisations confirmed the model's effectiveness in addressing defects. Therefore, these findings provide the confidence and rationale for the model's adoption by other similar organisations aiming to address defects in MDS development. Moreover, the findings of this research are relevant to other safety-critical software domains facing similar problems of defects.

3.9 Survey Procedure

The survey strategy uses a representative portion of the population from which the findings can later be generalised to the entire population (Glasow 2005). This approach provides the possibility of generating findings that are statistically representative of the study population at a lower cost (Saunders et al. 2019). While survey data is primarily collected through questionnaires, other means such as interviews and a mix of both questionnaire and interviews are also used (Ponto 2015). Surveys can be categorised as either qualitative or quantitative (Jansen 2010; Braun et al. 2021). A quantitative survey analyses the frequency of member characteristics within a population, while a qualitative survey examines the diversity of these characteristics (Jansen 2010). Quantitative surveys primarily employ closed questions to collect short answers such as yes or no. In contrast, qualitative surveys use basic and branching open-ended questions to collect long-form written responses from participants. Thus, qualitative surveys provide the potential to gain a deeper and new understanding of issues being investigated (Braun et al. 2021).

As part of answering research **RQ1** and **RQ2**, a questionnaire-based survey (Aithal and Aithal 2020) was conducted targeting participants in the MDS domain. This survey utilised a mix of both closed and open-ended questions to collect rich data. The objective was to gather first-hand information on whether agile practices are used to address defects in MDS development, and how they are implemented for this purpose.

3.9.1 Target Population

The survey conducted in this research targeted developers from development teams in MDS organisations in Ireland to gather first-hand information on which agile practices are used for addressing defects and their implementation. It was deemed important to gather data from practicing MDS developers as they are directly involved in addressing defects. However, identifying MDS organisations to be contacted for the survey was a challenge. The Irish Medical Devices Association's 2022-2025 strategy reveals that there are over 450 MedTech organisations in Ireland (Ibec 2022). However, it does not provide a detailed list of these organisations. In contrast, the 2016-2020 strategy categorises several MedTech organisations including 62 that develop MDS (Ibec 2016). This 2016-2020 strategy document was used as a starting point in identifying MDS organisations that were contacted for the survey. Additionally, MDS organisations with existing working relationships with the Regulated Software Research Centre were invited to participate in the survey. In total, 105 MedTech

organisations were contacted via phone or email to determine if they develop MDS and if they could participate in the survey. Through this process, 85 organisations were identified. However, only 65 of these organisations indicated their willingness to participate in the survey. Subsequently, an information leaflet, a consent form, and a link to the survey were emailed to 65 developers from different MDS development organisations.

Fifteen responses were received from different organisations, including twelve SMEs and three large organisations, as per the European Union's definition of Small and Medium-sized Enterprises (SMEs) (EU 2022). The received responses were coded, and an excel worksheet of the survey responses was exported from the Microsoft Forms tool. Each of the closed questions was then individually considered and analysed in Microsoft excel. The open-ended questions were manually analysed as the responses to these were few. Survey results are presented in section 4.5.1.

3.9.2 Sampling Technique

There are two main sampling techniques in research, probability sampling and non-probability sampling (Taherdoost 2016). Probability sampling guarantees every individual in the population has an equal opportunity to be selected for inclusion in the sample. It includes techniques such as simple random, stratified random, and cluster sampling (Taherdoost 2016). In contrast, non-probability sampling involves the researcher choosing participants based on criteria like availability, geographical proximity, or expertise considering the purpose of research (Ayhan 2011). It includes techniques such as quota sampling, purposive/judgement, snowball, and convenient sampling (Taherdoost 2016; Stratton 2023). The survey conducted in this research employed non-probabilistic purposive sampling technique. According to Robinson (2014), purposive sampling involves the intentional selection of informants based on their ability to elucidate a specific theme, concept, or phenomenon. This method was deemed appropriate as the survey targeted specific participants (MDS developers) to obtain first-hand information on the agile practices being used for addressing defects during MDS development.

3.9.3 Sample Size for Survey

The literature indicates that in qualitative research, there are no fixed rules for determining sample size. The appropriate sample size depends on several factors, including the research question, the purpose of the study, the stakes involved, the credibility of findings, nature of the topic, and the time and resources available for the study (Marshall et al. 2013; Braun et al.

2021). The sample size for the survey conducted in this research, might have been influenced by the willingness of MDS organisations to participate and the developers' time constraints due to work deadlines. Additionally, the time constraint for this academic study might have affected the sample size. Considering these factors, it was determined that a minimum sample size of ten experienced MDS developers from different MDS organisations was appropriate to achieve the research objective. After distributing the survey, fifteen responses were received and considered for analysis. While a large sample size is ideal for generalising findings, it requires collecting and analysing a large volume of data within a limited time. This survey focused on gathering insights into the use of agile practices for addressing defects in MDS development and their implementation, rather than generalising findings. Consequently, a key consideration in this research was the quality of the data, which is determined by its depth, richness, and appropriateness in achieving the research objectives (Stenfors et al. 2020). A well-designed questionnaire and its accompanying questions play a critical role in collecting high-quality and rich data (Hyman and Sierra 2016). In this research, guidelines proposed by Harlacher (2016) were followed in developing the questionnaire to ensure quality of the survey data. This process is detailed in section 3.9.6.

3.9.4 Ethical Clearance

In compliance with the Research Ethics Policy of Dundalk Institute of Technology, ethical approval was obtained before engaging any stakeholders in this research. The approval process required the submission of four documents to the Research Ethics Committee within the School of Informatics and Creative Arts. The documents included the ethics application form, participant consent form, participant information leaflet, and the survey questionnaire. These documents were submitted on October 13, 2023, and ethical clearance was granted on October 26, 2023.

3.9.5 Survey Method

There are different types of survey methods for data collection, such as verbal surveys (telephone and in-person interviews) and online surveys (Avedian 2014; Glasow 2005). Verbal surveys can be labour-intensive, expensive, and time-consuming, requiring a lot of preparation, coordination, and effort from both researchers and participants (Mathers et al. 2009). In contrast, online surveys are more flexible and can be distributed via various electronic means, which significantly facilitates the collection of rich data (Tanner 2018; Braun et al. 2021). Additionally, online surveys are a cost-effective means of collecting data from a large

population (Avedian 2014). Considering these factors and the time constraint for this academic study, the online survey method was considered as most appropriate. Participants from MDS organisations were emailed a link to the survey created using the Microsoft Forms tool, which is approved by DkIT. Additionally, an information leaflet and consent form were emailed to respondents.

3.9.6 Development and Validation of Survey Instrument

The survey questionnaire used in this research was developed following guidelines proposed by Harlacher (2016). These guidelines include, 1) determining the goal or goals of the questionnaire, 2) defining the information needed to address each goal, 3) writing the questions, 4) reviewing the questions for alignment with the goals and adherence to research-based guidelines for writing questions, 5) organising and formatting the questionnaire.

- *Step 1: determining the goal or goals of the questionnaire*

The overarching goal of the survey in this research was to gather data on the use of agile practices for addressing defects in MDS development. The aim was to reinforce and validate the findings from the literature review and to gather insights useful for answering part of RQ3: *How can an agile-based model better be designed to assist MDS developers to address defects*.

- *Step 2: defining the information needed to address each goal*

In this step, relevant information in relation to the goal of the survey was generated as follows:

1) Identifying agile practices that are used for addressing defects in MDS development
2) Understanding how agile practices are applied to address defects in MDS development (defect prevention and identification).

- *Step 3: writing the questions*

This step involved the researcher writing both close-ended and open-ended questions and reviewing them to ensure they were unambiguous and had only one intended meaning (Harlacher 2016). These questions were then included in the initial draft of the questionnaire.

- *Step 4: reviewing the questions for alignment with the goals and adherence to research-based guidelines for writing questions*

Step 4 entailed reviewing the questionnaire for content validity and pre-testing it with five members from the Regulated Software Research Centre at DkIT. The focus was to obtain feedback concerning the overall quality of the questionnaire's construction and their flow.

- *Step 5: organising and formatting the questionnaire*

This step involved refining the questionnaire based on the feedback received from the review and pre-testing processes and preparing a final version for implementation.

The final questionnaire comprised four sections. The first section collected general information including respondent's organisation profile (size and experience), respondent's experience in MDS development, their position in the current organisation, and the software development team size. The subsequent sections focused on software development methodologies and their use in the organisation, as well as measures used to address defects. The final section assessed the organisation's use of agile practices specifically for addressing defects in MDS development. The questionnaire utilised a mix of close-end and open-ended questions, with a majority being close-ended questions to facilitate easier and quicker completion (Hyman and Sierra 2016). Open-ended questions were included to gather deeper insights into measures for addressing defects and the application of agile practices for this purpose.

Subsequently, the survey link, consent form and an information leaflet were emailed to respondents. To increase the response rate, follow up emails were sent to non-respondents every fortnight, twice. Finally, a thank-you email was sent to all 65 organisations, expressing gratitude for their participation and reminding those who had not yet responded that they could still do so.

3.10 Semi-structured Interviews

3.10.1 Introduction

There are numerous methods for gathering data in research. The choice of a particular method depends on several factors such as the research objectives and the available resources. Among the many data collection tools, interviews are the most commonly and widely utilised (Alshenqeeti 2014; Jamshed 2014). Interviews can be of three different forms: unstructured, structured, and semi-structured. Structured interviews involve predetermined questions that typically require Yes or No responses. These interviews do not offer the flexibility for the interviewer to probe further, or the interviewee to expand on their responses. Unlike structured

interviews, unstructured interviews do not typically employ preset questions. Thus, they allow greater latitude for interviewees to express their responses in their own ways and at their own pace (Jamshed 2014). Additionally, interviewees have the freedom to ask questions and interviewers to provide their opinions. Nevertheless, the interviewer exerts minimal control to ensure the research objectives are achieved.

Semi-structured interviews are characterised by their flexible structure. Typically, these interviews are structured using an interview guide that highlights the main focus areas, topics or themes to be discussed (Adams 2015). The interview guide is generally designed to allow flexibility in both how and what order questions are approached during the interview. Additionally, the interview guide facilitates the follow up and further exploration of certain topics or questions (Priyadarshini 2020).

To gather data about the efficacy and suitability of the AV4AD model to address defects in MDS development, semi-structured interviews were conducted with developers in MDS organisations in Ireland. These interviews aimed to answer part of RQ3:

- *How can agile practices be integrated in MDS development to address defects?*

Semi- structured interviews were chosen for this part of the research because of their ability to provide in-depth information from individuals that have experience in the study domain (DeJonckheere and Vaughn 2019). Semi-structured interviews comprise a mix of closed and open-ended questions that are supplemented by follow up questions designed to probe or prompt, to gather information from an individual respondent (Adams 2015). This technique was considered suitable for this research because the purpose was to gain in-depth information about the implementation of the AV4AD model to address defects in MDS development and its improvements. Semi-structured interviews were vital in this research for gathering opinions, perspectives, ideas and experiences of MDS developers about the efficacy and suitability of the AV4AD model for addressing defects. DeJonckheere and Vaughn (2019) state as follows:

> *"The overall purpose of using semi-structured interviews for data collection is to gather information from key informants who have personal experiences, attitudes, perceptions, and beliefs related to the topic of interest".*

3.10.2 Developing the Interview Guide and Questions

This part of the research commenced with the creation of the interview guide, indicating key focus areas for the semi-structured interview. The interview guide is an important tool for researchers during interviews. It consists of a list of questions which directs conversation towards the research topic during the interview (Kallio et al. 2016). It ensures that all important questions are addressed and provides a consistent flow of key questions for all interviewees (Adams 2015). This in turn not only prevents the omission of important questions but also aids in the subsequent analysis of the collected data. Additionally, the interview guide helps the researcher to maintain focus on the particular theme during the interview. In developing the interview guide for this research, guidelines proposed by Kallio et al. (2016) were followed. These include five stages as follows: 1) identifying the prerequisites for using semi-structured interviews, 2) retrieving and using previous knowledge, 3) formulating the preliminary semi-structured interview guide, 4) pilot testing the guide, and 5) presenting the complete semi-structured interview guide.

Kallio et al. (2016) point out that the aim of the first stage is for the researcher to determine the suitability of utilising semi-structured interviews as a data collection tool in relation to the research questions. Accordingly, semi-structured interviews were deemed appropriate for this part of the research to gather opinions, perspectives, ideas and experiences of MDS developers about the efficacy and suitability of the AV4AD model for addressing defects and its potential improvements. This aims to address part of research question (RQ3): *How can agile practices be integrated in MDS development to address defects?* The second stage involves the researcher gaining understanding of the subject by appraising existing knowledge (Kallio et al. 2016). In this research an extensive literature review, detailed in Chapter 2 of this thesis, and a survey of MDS developers, detailed in Chapter 4, were carried out to identify the need for the AV4AD model. These processes informed the evaluation of the AV4AD model through semi-structured interviews, allowing MDS developers to provide feedback on how it could address defects.

The third stage is the formulation of the interview guide as a tool for gathering in-depth data during the interview. One of the key parts of the interview guide are well-formed interview questions that ensure relevant data is collected to achieve research objectives. However, formulating interview questions that yield rich data can be challenging. Therefore, for researchers to generate relevant data, it is important that considerable attention is given to the

development of interview questions. Interview questions can be categorised into two main forms: Open-ended and Closed. Closed questions are characterised by typically short or one-word answers such as Yes or No. In contrast, open-ended questions provide greater latitude for the interviewee to elaborate on the researcher's topics of interest (Roulston 2023). The responses to open-ended questions are typically detailed descriptions which can be further probed for clarity or additional information. This research adopted open-ended questions that afforded the interviewees the flexibility to provide more elaborate responses during the conversation. Initial probe questions were included where necessary, but these were mainly generated during the conversation from interviewee's provided responses.

The fourth stage is pilot testing of the interview guide which aims to evaluate the relevance of the content of the preliminary interview guide (Kallio et al. 2016). This is an important aspect after developing the interview guide to assess its efficacy in capturing the data the researcher aims to collect (Malmqvist et al. 2019). In this research, the interview guide was iteratively piloted with five members from the Regulated Software Research Centre within the School of Informatics and Creative Arts at DkIT. The purpose of these pilot interviews was to eliminate ambiguities in the questions, assess the sufficiency of the proposed duration for the interview, and to evaluate the relevance and flow of questions. The interview guide was subsequently revised and refined based on the feedback obtained from the pilot interviews before the actual interviews. Overall, the pilot interviews contributed to the accuracy and consistency of the interview guide. The fifth and final stage is presenting the complete interview guide. The complete interview guide developed by following the four stages detailed in this section is presented in Appendix D. The interview guide was utilised in collecting data for this part of the research.

3.10.3 Sample Size for Semi-structured Interviews

Determining sample size when conducting qualitative research like semi-structured interviews is a widely known problem (Squire et al. 2024; Nakkeeran 2016; Bekele and Ago 2022). Due to the uncertainty in the number of respondents available to be interviewed, determining the exact sample size for semi-structured interviews is difficult. Qualitative researchers argue that when the goal is to gather in-depth knowledge, a strict focus on sample size cap is not necessary (Moser and Korstjens 2018; Nakkeeran 2016; Hong and Cross Francis 2020; Boddy 2016). Instead, researchers should focus on the quality and richness of the collected data to meaningfully address the research question (Malterud et al. 2016; Moser and Korstjens 2018).

However, to ensure data reliability, qualitative researchers recommend including at least five sources (Tongco 2007; Hennink and Kaiser 2022). Consequently, a sample size of 8 interviewees was deemed adequate for this part of the research. The focus was on gaining in-depth insights on how the AV4AD model could assist MDS organisations in addressing defects, rather than collecting broad perspectives. A sample size of 8 interviewees was also considered appropriate for this research to achieve meaningful, timely, and qualitative analysis, which can often be hindered by the use of larger sample sizes (Boddy 2016).

3.10.4 Selecting Interviewees

Selecting respondents for semi-structured interviews is a key aspect in obtaining the best information (DeJonckheere and Vaughn 2019). MDS developers were considered for this part of the research because they are directly involved in MDS development and addressing defects. They were deemed suitable to provide solid opinions about the AV4AD model because of their lived experiences and knowledge in MDS development. Consequently, purposive sampling was employed for this research. Purposive sampling is a widely used non-probability method for collecting quality and information-rich data to meet the objectives of qualitative research (Palinkas et al. 2015). From the survey conducted with developers in MDS organisations, eleven respondents indicated their willingness to further participate in the research. This information was valuable for this part of research. All the eleven respondents were selected for participation in the semi-structured interviews. A formal email request was sent to all the eleven respondents, including the alpha version of the AV4AD model and a document providing the information about the AV4AD model. Respondents were requested to conveniently indicate their availability using the Booking app in Microsoft Teams via a link that was shared to them in the email. This approach helped to properly manage the interview schedules by preventing overlaps in time slots. Through this process, six respondents confirmed their availability and indicated their convenient times for the semi-structured interviews. Further efforts were made in which 50 organisations that did not participate in the survey were invited through email to participate in the semi-structured interviews. Through this process, two additional participants from different SMEs that have been developing MDS for more than three years were identified, bringing the total number of participants to eight. As mentioned previously, this number of respondents was deemed suitable for this part of the research considering the purpose and time required to conduct the semi-structured interviews.

3.10.5 Profiles of Interviewees

Involving the right subjects is crucial for ensuring that collected data aligns with the research purpose. This not only enhances the relevance of the data but also the appropriate use of resources. The interviewees included seven software developers or engineers and one who also worked as a quality and regulatory specialist. All interviewees have over ten years of experience in software development with a minimum of five years in MDS development.

One of the challenges in collecting data through semi-structured interviews when respondents are exclusively from either SMEs or large organisations is the potential for bias. This can lead to conclusions that are not representative of both SMEs and large organisations. To overcome this, it is important to have views from both groups. It is difficult to achieve this balance as researchers cannot be certain about having an equal number of respondents from both groups. Given that participation in this research was voluntary, a strict focus on equal representation was not necessary. Instead, the primary objective was to gather quality and rich data on the efficacy and suitability of the AV4AD model in addressing defects, regardless of organisation size. Nonetheless, the research gathered perspectives from both SMEs and large organisations. Six interviewees were from SMEs and two from large organisations. The two large organisations have been developing MDS for over 20 years. One of the three SMEs has also been developing MDS for over 20 years, two for over 10 years, and three for more than 3 years.

3.10.6 Conducting the Semi-structured Interviews

The eight interviews were conducted separately, scheduled on different days at times convenient for each respondent through the Microsoft Booking app. The interviews were conducted online using Microsoft Teams application. One of the interviews included a software developer and a quality assurance specialist from the same organisation. Their combined participation enriched the data as each was able to complement and extend the other's views on the AV4AD model during the discussion. At the beginning of the interviews, respondents were informed of their anonymity in the aggregated findings in the report. They were further asked about whether they would allow the researcher to record the discussion. Each of the eight respondents gave consent to record the interview. The interviews were recorded using the recording facility within Microsoft Teams application and saved to Microsoft OneDrive. Copies of each of the eight interview recordings were also stored locally on the researcher's laptop and external disk drive. To maintain anonymity and confidentially, the recordings were stored using pseudonyms like 'Interviewee 1' etc. The names of respondents and their

organisations were only used by the researcher when follow up was needed for further clarification on the collected data. On average each of the eight interviews took approximately seventy minutes resulting in a total of 155 pages of transcription text. The semi-structured interview transcripts can be found in Appendix E.

3.10.7 Transcribing the Interviews

Accurate transcription of interview recordings is key for extracting important insights and achieving the objectives of the interviews. A transcription of each interview was obtained from the Microsoft Teams app immediately after its conclusion and carefully reviewed alongside the interview record. This was done to ensure data accuracy. After reviewing the first transcript, it was observed that significant portions of the interview were not transcribed, and numerous errors were identified. Consequently, a decision was made not to rely on the Teams app transcription for the analysis. Utilising a combination of both verbatim and reconstructive transcription approaches (Brinkmann 2023), the researcher manually transcribed all the eight interviews to ensure accuracy and completeness of the data. Additionally, the researcher maintained context and meaning of the interviews, which is important for the trustworthiness of the data (Malmqvist et al. 2019). The completed transcripts were subsequently shared with the interviewees for their review and correction of their answers. Additionally, they were invited to provide feedback to the researcher on any items that might have been incorrectly captured. Each interviewee confirmed that the transcript captured what they communicated. The semi-structured interview transcripts can be found in Appendix E.

3.11 Summary

This chapter has presented the research methodology suitable for this research by reviewing various components that constitute it. The components of the research methodology for this study are presented in table 3.1. The philosophical perspective chosen for this study is pragmatism and both qualitative and quantitative approaches are employed. The early stage of this research involves an inductive approach where data is gathered about agile practices to build a theory about whether they could be useful to address defects in MDS development. In the later stage of this research a deductive approach is employed to ascertain the use of identified agile practices to address defects.

This chapter has demonstrated how this study aligns with the DSR and identified the DSRM as the overarching research strategy utilised to address the research questions and accomplish

the research objectives identified in this research. Unlike ADR, DSRM does not assume any specific client nor collaboration with a client and the activities are carried out sequentially. Additionally, DSRM aims to build artefacts that are utile for any client. This research aligns with the DSRM as it builds a solution for any MDS organisation to address defects. The DSRM is a form of DSR which has its foundation in the pragmatism. DSRM incorporates principles, practices, and procedures to carry out DSR and facilitates multiple entry points in the development process (Peffers et al. 2007). Chapter 3 has also detailed the methods undertaken to ensure the reliability and validity of this research. Furthermore, it has provided detailed accounts of the survey and semi-structured interview processes followed in this research.

Table 3.1: Summary of research methodology

Philosophy	Pragmatism
Approach	Mixed approach (deductive and inductive)
Strategy	DSRM
Choice	Mixed-method (quantitative and qualitative)
Time horizon	Cross-sectional

4 Development of the AV4AD Model

4.1 Introduction

This chapter presents the development of the AV4AD model. The chapter begins by presenting a detailed analysis, focusing on the evidence of using each identified agile practice in section 2.8, to address defects in the GSD and its suitability to address defects in MDS development. It then provides a discussion on the analysis of agile practices, followed by an assessment of agile practices usage for addressing defects in GSD and MDS development. The chapter then presents survey results and analysis. The survey was conducted to gather agile practices that are currently used in MDS development, specifically focusing on whether they are used to address defects and insights into their application for this purpose. The chapter also presents the proposed solution, detailing its development and recommended application for best results in addressing defects. The chapter concludes with a summary of the key aspects discussed.

4.2 Analysis of Agile Practices

This section presents the analysis conducted to uncover evidence of using agile practices to address defects in the GSD and determining their suitability to address defects in MDS development. To gather all relevant studies, an extensive search for literature was necessary. Several search statements were constructed and used to find relevant studies in various online sources. These statements included interchangeable terms defect, fault, bug, error, flaw, failure, mistake, and anomaly. Each term was searched individually within each statement. This process ensured that relevant studies on each agile practice were identified and not missed. Additionally, reference lists of the identified papers were examined to uncover other relevant studies. The following are examples of search statements for studies on PP.

1. Pair programming and software defects/errors/faults/bugs/failures/flaws/anomalies/mistakes
2. Using pair programming to detect/identify/reduce/remove software defects/errors/faults/bugs/failures/flaws/anomalies/mistakes
3. Case study of using pair programming to detect/identify/reduce/remove software defects/errors/faults/bugs/failures/flaws/anomalies/mistakes
4. Using pair programming to prevent software defects/errors/faults/bugs/failures/flaws/anomalies/mistakes

5. Detecting/identifying software defects/errors/faults/bugs/failures/flaws/anomalies/ mistakes using pair programming.
6. Effect/impact of pair programming to detect/reduce software defects/errors/faults/ bugs/failures/flaws/anomalies/mistakes.

All the retrieved papers containing relevant terms were listed in Microsoft Excel and were grouped in different worksheets based on the agile practice. Studies discussing the use of agile practices or using agile practices specifically to address these terms were analysed. Studies not discussing the use of agile practices were discounted. In total, 667 relevant papers were found, reviewed and analysed for this part of the research. The findings on each agile practice are presented in the following subsections. Table 4.1 provides the definitions of key terms used in subsequent tables presenting the results on agile practices.

Table 4.1: Definition of terms used in subsequent tables

Term	**Definition**
Improvement	The application of the agile practice resulted in defect reduction compared with instances where the practice was not applied.
No improvement	The application of the agile practice did not result in defect reduction.
No difference	The results of applying the agile practice were equivalent to those of a comparison technique.
Experience report	Presents the author's first-hand account that offers new insights or reflections into a specific phenomenon such as the description of a new approach.

Findings on Pair Programming

Out of 667 studies analysed, 105 were on PP and 18 of these discussed it in relation to software defects. A summary of the 18 papers is presented in table 4.1. No study was found from the MDS domain that discussed using PP to address defects. An experience report presents the author's first-hand account that offers new insights or reflections into a specific phenomenon such as the description of a new approach (Oliveira Lopes 2012; Mancl and Fraser 2021). As shown in table 4.2, of the 18 studies, 16 report improvement in using PP to address defects.

Table 4.2: Summary of PP studies on software defects

Study type	Improvement	No improvement	No difference
Case study	5	0	1
Experiment	5	1	0
Experience report	5	0	0
Survey	1	0	0
Total	**16**	**1**	**1**

Sixteen of the 18 studies reported that PP significantly reduced defects and lowered the introduction of new defects in the code. The details of these studies can be found in Appendix B.1. One experimental study found no improvement in defect reduction, while a case study reported no difference in defect reduction. Hulkko and Abrahamsson (2005) reported conflicting results, while Madeyski (2008) attributed the result to small project size. The findings show that PP prevents defects in software development when applied correctly. Working with a partner increases the likelihood of early defect detection as code is continuously examined during development. In PP, active communication is emphasised, allowing programmers to discuss and understand requirements, which increases the likelihood of correctly implementing solutions. PP can also be combined with other practices such as TDD and code review for best results in preventing defects. Overall, the benefits of using PP to address defects outweigh any potential drawback. Given these findings and the fact that PP has been used in MDS development, as outlined in Chapter 2, section 2.8, it is recommended for addressing defects in MDS development. It is essential that programmers involved in PP should be adequately skilled to identify defects and avoid introducing new ones as they take turns writing code.

While PP is traditionally used during coding, this research recommends a form of PP during the early stages of the development lifecycle of MDS. This approach involves developing unit tests and test scenarios early which can assist in detecting defects early and ultimately preventing their occurrence later. Involving PP in the early stages also helps in clearing any ambiguities that could lead to incorrect implementation and defects later. For example, during software requirements analysis, a pair of testers could work together to develop unit tests that would later be used to verify correct implementation of a functionality.

Findings on Test-Driven Development

Out of 667 studies analysed, 100 were on TDD and 38 of these reported its effect on software defects. It is interesting that while TDD is advocated to reduce defects, no study was found from the MDS domain that discussed using it to address defects. Most of the 38 studies compared TDD to test-last approach, while one compared TDD to code inspection. Table 4.3 summarises the 38 studies. Of the 38 studies, 34 report TDD reduced defects.

Table 4.3: Summary of TDD studies on software defects

Study type	**Improvement**	**No improvement**	**No difference**
Experience report	10	0	0
Experiment	9	2	2
Survey	2	0	0
Case study	8	0	0
Experiment & survey	4	0	0
Focus group	1	0	0
Total	**34**	**2**	**2**

Thirty-four studies found TDD effectively reduced defects. The details of these studies can be found in Appendix B.1. Two studies found no difference compared to a test-last approach (Pančur and Ciglarič 2011; Erdogmus et al. 2005). One study found code inspection better at reducing defects than TDD, but more costly (Wilkerson et al. 2012). Another study found test-last development more effective at reducing defects than TDD (Munir et al. 2014). The findings reveal that TDD effectively addresses software defects. By writing tests first and running them frequently during development, developers can find and remove defects early on, improving software quality and preventing costly rework later. Using TDD to convert requirements into test cases, can help developers gain a clearer understanding of the requirements and reduce the likelihood of defects caused by ambiguity. Writing tests first allows developers to think and through communication with stakeholders, gain a clearer understanding beforehand about the purpose of the functionality to be implemented. TDD can also be used in combination with other practices such as PP and onsite customer, guaranteeing more effective defect detection and removal. Given its success to address defects in GSD and the fact that it has been used in MDS development, as outlined in section 2.8, TDD is recommended for addressing defects in MDS development.

While TDD is traditionally associated with the coding stage, it can be more useful for preventing defects during the early stages of software development lifecycle. This research recommends the application of TDD during requirements analysis stage of MDS development lifecycle for translating user stories into testable requirements or scenarios and writing tests. This approach ensures that defects that arise due to ambiguities are prevented early.

Findings on Unit Testing

Out of 667 studies analysed, 43 were on unit testing and 26 of these discussed its use in addressing software defects. Table 4.4 summarises the 26 studies.

Table 4.4: Summary of Unit Testing studies on software defects

Study type	**Improvement**	**No improvement**	**No difference**
Experience report	2	0	0
Experiment	13	3	0
Survey	1	0	0
Case study & Survey	1	1	0
Case study	4	1	0
Total	**21**	**5**	**0**

Twenty-one studies reported improvements in defect detection and removal. These studies are available in Appendix B.1. The remaining five studies found low defect detection effectiveness for unit testing. These studies can be found in Appendix B.2. The findings show that unit testing addresses defects in individual software components before they are integrated into a larger system. By testing smaller units of code, developers can more easily detect and fix defects before integration where they can be more difficult to resolve. Additionally, writing unit tests early, based on requirements, can help developers better understand the requirements and reduce the likelihood of defects caused by ambiguity. As the software is being developed, developers can use these unit tests to verify its functionality. Based on the findings in this section, and the fact that unit testing has been used in MDS development, as outlined in section 2.8, it is recommended as a means of identifying defects in MDS development.

Findings on Refactoring

Out of 667 studies analysed, 72 were on refactoring and 23 of these discussed its use in addressing software defects. No study was found from the MDS domain that discussed using refactoring to address defects. The 23 studies are summarised in table 4.5 and shows 22 report an improvement.

Table 4.5: Summary of Refactoring studies on software defects

Study type	**Improvement**	**No improvement**	**No difference**
Case study	7	1	0
Experiment	13	0	0
Case study, Survey & Interview	2	0	0
Total	**22**	**1**	**0**

Of the 23 studies, 22 reported decreases in defects after refactoring and that refactoring reduced defect proneness in code. The remaining study found that only 15% of defects were likely induced by refactoring actions. These studies can be found in Appendix B.1. The findings show that proper and complete refactoring reduces defects in software. However, it is crucial that refactoring should be followed by appropriate testing such as unit testing and integration testing, depending on the type and amount of refactoring done. This addresses any defects that may potentially be introduced into the code, thereby preserving the quality of software. The choice of the refactoring type should be critically considered as it can impact the outcome of the refactoring activity. It is also essential to determine the optimal amount of refactoring and when to stop, because excessive refactoring may induce defects in the software. Refactoring can be used in combination with PP to help prevent any potential mistakes that may impact the quality of software. Based on the findings presented in this section and its adoption in MDS development, as outlined in section 2.8, refactoring is recommended for addressing defects in MDS development. Also, considering the long lifecycle of medical devices and the need for ongoing software updates and maintenance, refactoring is recommended to support these processes to prevent introducing defects and identifying those that might have escaped the testing process.

Findings on Code Review

Out of 667 studies analysed, 43 were on code review and 19 of these discussed its use in addressing software defects. No study was found from the MDS domain that discussed using code review to address defects. A summary of the 19 studies is presented in table 4.6 and shows 18 report improvement in addressing defects using code review.

Table 4.6: Summary of Code Review studies on software defects

Study type	**Improvement**	**No improvement**	**No difference**
Case study	12	1	0
Case study, Survey & Interview	2	0	0
Case study & Interview	1	0	0
Experience report	2	0	0
Experiment & survey	1	0	0
Total	**18**	**1**	**0**

Out of the 19 studies, 18 reported that code review identified defects early in the development process, reduced software's proneness to defects, and that best results can be achieved when it is performed by people with specific skills. One study found that 54% of reviewed changes introduced bugs in the code and linked this result to personal and participation metrics, such as the number of developers involved, that impact code review quality. These studies can be found in Appendix B.1. The findings show that code review addresses software defects. To achieve the best results, code reviews should be performed by skilled developers. Additionally, factors such as reviewer workload and the number of reviewers involved should be considered for more effective defect identification and removal. Code review can be more beneficial if applied early on and regularly during the development process than waiting until large amount of code is written. Apart from reviewing of the implementation code, review of test code against the specified functional requirement is essential to ascertain that it is defect-free and is testing the correct thing. Despite the limited research on its use in MDS development, as outlined in chapter 2, section 2.8, code review is recommended as a means of addressing defects in MDS development, based on its proven success for this purpose in the GSD.

Findings on Continuous Integration

Out of 667 studies analysed, 32 were on CI and 10 of these discussed its use in addressing software defects in GSD, but none was found from the MDS domain. All 10 studies reported improvement using CI to address defects. The 10 studies are summarised in Table 4.7 and discussion follows. All the ten studies reported that CI increased defect detection and fixing early, reducing customer-reported defects. These studies can be found in Appendix B.1. Vassallo (2019) also noted that bad practices by developers when implementing CI can result in build failures, suggesting that the effectiveness of CI depends on the correct and rigorous application of the CI process.

Table 4.7: Summary of CI studies on software defects

Study type	**Improvement**	**No improvement**	**No difference**
Experience report	1	0	0
Experiment	1	0	0
Survey	1	0	0
Case study	3	0	0
Case study & Interview	2	0	0
Survey & Interview	0	0	0
Case study & Survey	2	0	0
Total	**10**	**0**	**0**

The findings show that CI addresses software defects. By making small, frequent changes to code and verifying them each time they are checked into the codebase, ensures developers can catch and fix integration and regression defects early. This prevents costly rework and improves software quality. CI can be used together with other practices such as TDD for even more effective results in defect identification. Based on these findings and its adoption in MDS development, as outlined in chapter 2, section 2.8, CI is recommended for identifying defects in MDS development.

Findings on Onsite Customer

Out of 667 studies analysed, 23 were on onsite customer and three of these discussed its use in addressing software defects. All the three studies demonstrate an improvement in addressing defects. A summary of these studies is shown in Appendix B.3. A study from the MDS domain

reported a significant improvement in software defect identification early in the development process due to user involvement in the development of a clinical communication system at Trillium Health Partners in Canada (Tang et al. 2018). Various strategies of user involvement were used at different phases including user-centred design, user co-design, and participatory design. Korkala et al. (2006) conducted four case studies exploring the impact of customer communication and feedback on software defect rates. They found that in one case study, 62.6% of defects were caused by lack of efficient customer communication and feedback. Kabbedijk et al. (2009) conducted a case study at AFAS ERP Software BV. They discuss various methods to involve customers in requirements gathering for future releases, including involvement through incident reports, idea feedback, suggestions, and role of the moderator. About 15% of incidents were software defects converted into product development requests to become future requirements, suggesting the importance of customer involvement in identifying defects and improving later releases.

The findings demonstrate the significance of customer involvement in software development to ensure that correct requirements are gathered. Customer feedback during software development plays a crucial role in defects identification and removal early on, preventing costly rework at later stages. Based on the findings presented in this section and the adoption of onsite customer in MDS development, as outlined in section 2.8, it is recommended as a means of preventing and identifying defects in MDS development. Establishing a close connection with customers or users is essential in software development, as it enables valuable feedback that can enhance the overall quality of the software.

Findings on Integration Testing

Integration testing has been widely researched for its primary goal of ensuring software components interact defect-free. Many studies have focused on developing approaches for implementing integration testing (Maciel et al. 2010; Akbari et al. 2017; Benz 2007; Jiang et al. 2021; Singh and Akhtar Khan 2012; Braspenning et al. 2006; Da Silva and De Lemos 2011). While there is general agreement that integration testing is a key practice for catching defects that escape individual component testing, few academic studies have specifically discussed its use for this purpose. Of the 25 studies analysed, only three reported results of integration testing and defect detection. These studies reported mixed results, with one finding that unit tests detected more defects than integration tests but required more time to fix (Orellana et al. 2017), another finding no significant difference in defect detection capability between the two

(Trautsch et al. 2020), and a third proposing a tool that re-uses unit tests to target specific defects during integration testing, demonstrating effectiveness in detecting targeted defects (Holling et al. 2016).

Despite the limited studies specifically focusing on the relationship between integration testing and defect reduction, this research recommends it for identifying defects in MDS development. Integration testing can help to catch defects that escape individual component testing and to prevent failures that may occur during the integration of MDS with hardware medical devices. Integration testing has also been used in MDS development, as outlined in section 2.8.

Findings on Behaviour-Driven Development

Out of 667 studies analysed, 15 were on BDD and four of these discussed its use in addressing software defects. Wang and Wagner (2018) found no significant difference between BDD and manual UAT in fault detection using a mutation score indicator, attributing the result to the simple code used in the experiment. Ali (2019) reported that BDD enabled efficient testing of mobile apps, tracking regression errors, and diagnosing localisation problems of right-to-left languages (RTL). Irshad et al. (2021) reported that a BDD-based process improved communication, quality, reuse, and documentation, but its application was time-consuming and required long-term commitment. Tavares et al. (2010) reported on using a BDD tool that emphasised specification and validation of anticipated behaviour, increasing the potential to produce code with fewer defects at functional and unit level. A summary of these studies is provided in Appendix B.3. The findings suggest that BDD can address defects if applied early in the development process. By using BDD to correctly gather customer requirements and write tests based on them, developers can ensure that the implemented behaviour meets the specified requirements. This ultimately guarantees that the written tests will pass when implementing and refactoring code. Also, by using BDD during coding, developers can improve code quality, design, maintainability, and reduce defects (Smart 2014). Despite the limited evidence, BDD is recommended for preventing defects in MDS development. However, it is important to carefully understand its process before implementation.

Findings on User Story

Despite being widely researched, few academic studies discuss utilising user stories to address defects. Out of 667 studies analysed, 17 were on user story and two of these discussed its use in addressing software defects. Scott et al. (2021) found that changes in number of defects were

associated with changes in user story quality scores, suggesting that quality user story helps to reduce software defects. Lucassen et al. (2016) showed that using user stories, a template, and quality guidelines such as the INVEST, improved productivity and work deliverables quality. However, the study did not explicitly address the effect of user story on software defects. INVEST stands for Independent, Negotiable, Valuable, Estimable, Small, and Testable. Considering the purpose of a user story to describe how a software feature will deliver value to the client, it can be useful for preventing defects, but its effectiveness depends on how accurately it is written and implemented. To maximise its capability, a user story should be employed in combination with other development practices such as PP, and TDD. Its effect is mainly on ensuring that requirements are correctly defined (Scott et al. 2021), enabling developers to build the right software product, preventing costly defects later in the development life cycle (Lucassen et al. 2016). Therefore, user story is recommended for preventing defects in MDS development.

Findings on Retrospective

Although retrospective is a widely advocated and researched practice in ASD, to date no study has been found that discusses using it specifically to address software defects. Out of 667 studies analysed, 15 were on retrospective. These studies discussed a range of other aspects such as effectiveness of using cause-effect diagram (CED) (Lehtinen et al. 2015), team learning and improvement (Dingsøyr et al. 2018), supporting Root Cause Analysis (RCA) retrospectives (Lehtinen et al. 2014), and issues discussed during retrospective meetings (Lehtinen et al. 2017). Considering that both CED and RCA are aimed at preventing recurrence of a problem or defect in the future (Dalal and Chhillar 2013), the association of retrospective with these techniques in the literature suggests that retrospective may serve a supporting role, rather than directly contributing to defect prevention and identification. Additionally, the lack of academic literature on retrospective in relation to addressing defects may suggest that research has not been done or that the practice does not have impact on defect identification and removal. Despite its use in MDS development, retrospective is not recommended for addressing defects in MDS development.

Findings on Code Ownership

Out of 667 studies analysed, 21 were on code ownership and five of these discussed its use in addressing software defects. A summary of these studies is shown in Appendix B.3. Bird et al. (2010) found that measures of ownership, such as the number of low-expertise developers and

the proportion of ownership for the top owner, have a relationship with both pre-release defects and post-release failures. Bird et al. (2011) found that components with several minor contributors had more failures than other components and that higher code ownership resulted in fewer defects. Greiler et al. (2015) found that low code ownership correlated with the number of bugs fixed on file or directory level, indicating that higher code ownership reduces code defect proneness. Faragó et al. (2015) concluded that common code is more defect-prone than code developed by fewer developers, resulting in maintainability problems. Foucault et al. (2014) found that defect-proneness was more influenced by module sizes than code ownership.

The findings show that developers with low levels of expertise and large module sizes in code ownership are more likely to introduce defects into the code. In the context of MDS development, it is essential to implement code ownership in a way that ensures all developers assigned to a module have a clear understanding and high level of expertise. This can help minimise the likelihood of defects. Strong ownership, which limits changes to the module to developers with specific skills, may be the most suitable form of code ownership for MDS development. Based on the findings presented in this section and its adoption in MDS development, as outlined in section 2.8, code ownership is recommended for preventing defects in MDS development.

Findings on User Acceptance Testing

Out of 667 studies analysed, 21 were on UAT and three of these discussed it in relation to software defects. Jeeva Padmini et al. (2016) found that defect tracking increased with the implementation of UAT process. Hugo Santiago C. Pinto et al. (2018) found significant differences in the number of detected faults using the parallel and automated UAT. Sowri Raja Pillai and Rani Hemamalini (2022) found that the hybrid development approach had the highest number of defects, indicating effectiveness of UAT in detecting defects. A summary of studies on UAT can be found in Appendix C.3.

Despite limited studies on UAT, it was commonly used in studies investigating other agile practices such as PP and TDD because it assures customers that the developed system has the expected components and that they are fit for purpose (Pandit and Tahiliani 2015; Miller and Collins 2002). By identifying test scenarios or functionalities from requirements early and developing UAT test cases based on them, both testers/users and developers can gain a clearer understanding of the requirements. This helps minimise the likelihood of defects caused by

ambiguity. The UAT test cases can be used to verify that the software or module has been implemented correctly. Given these findings and its adoption in MDS development, as outlined in chapter 2, section 2.8, UAT is recommended for preventing and identifying defects in MDS development. To achieve the best results, it is advisable to implement UAT early and regularly in the development process rather than waiting until the final release. This provides early feedback to the development team, reducing time and rework costs by resolving the problems early before they escalate (Jeeva Padmini et al. 2016).

Findings on Daily Stand-up Meeting

Out of 667 studies analysed, 13 were on daily stand-up meeting and none discussed using it to address defects. These studies covered aspects such as member interactions (Stray et al. 2012), team attitudes towards stand-up meeting (Stray et al. 2016), and whether developers found them valuable (Stray et al. 2017). Common findings suggest that information sharing and problem-solving are the important aspects of daily stand-up meetings that contribute to project success. Given the lack of evidence supporting daily stand-up meetings in addressing software defects, it is not recommended for addressing defects in MDS development.

Findings on Rapid/Shorter/Small Releases or Frequent Builds

Out of 667 studies analysed, 28 were on small releases and five of these discussed it in relation to software defects. Four of the five studies reported that with small releases defects were discovered and fixed earlier, reducing post-release defects (Khomh et al. 2012; Souza et al. 2014; Abou Khalil et al. 2021; Abou Khalil et al. 2019). The fifth study reported that variable short releases did not have any significant effect on defect handling metrics (Jansen et al. 2021). A summary of studies on small releases is shown in Appendix B.3.

While the practice of shorter releases has primarily been implemented in GSD to address defects, it can also be useful in MDS development when used in conjunction with onsite customer. Developers could regularly demonstrate small pieces of functionality to the customer, who would then provide feedback to ensure that the correct behaviour is implemented. This can help to ensure defects are detected and fixed early before the software components become complex. Ultimately guaranteeing that the final product meets the needs of the client and regulatory requirements. Therefore, small release is recommended for addressing defects in MDS development.

Findings on Simple Design

Out of 667 studies analysed, 17 were on simple design and one of these discussed it in relation to software defects. The study reported that the increase in design flaws likely induced defects in software, suggesting that the more complex designs result in increased chances of post-release defects (D'Ambros et al. 2010). Academic literature on using simple design to address defects is scarce, but Jonsson et al. (2012) advocate that it can reinforce other practices such as PP, refactoring, coding standards, and collective code ownership. A simple design can allow developers to easily comprehend the software to be built and make modifications as needed with ease because it can serve as a clear blueprint. This can help to reduce the likelihood of defects that may occur due to misunderstanding of the requirements. Provided with a simple design, developers can more effectively address any problems that may arise during the development process of the software. Despite the limited evidence in relation to its use to address defects, simple design is recommended as a means of preventing defects in later stages of MDS development as any problems are spotted early in the design. This is achieved by enabling developers to easily comprehend system functionality through the creation of a simple design.

Findings on Coding Standards

Out of 667 studies analysed, 21 were on coding standards and five of these discussed it in relation to software defects. One study found a correlation between the violated coding rules and actual defects (Boogerd and Moonen 2008). Another study found that using an IDE and adhering to coding standards reduced errors and the likelihood of new faults (Popic et al. 2018). Dorin (2018) proposed a two layer coding standard to reduce faults and complexity. Capiluppi et al. (2009) found marginal improvement in some aspects of quality for KDE projects but no improvement in complexity. Zaidman (2004) found that programs produced after the implementation of coding standards were more reliable. A summary of studies on coding standards is shown in Appendix B.3. The findings show that coding standards can help to increase the reliability of software if implemented in combination with appropriate development practices such as PP. Considering the safety-critical nature of MDS, adherence to coding standards that enhance the likelihood of achieving reliability is essential. Therefore, coding standards is recommended for preventing defects in MDS development.

Findings on Use Case

Out of 667 studies analysed, 12 were on use case and none of these discussed it in relation to software defects. Most of these studies discussed quality of use cases to ensure they are defect-free. For example, Liu et al. (2014) proposed an advanced parsing technique to automatically detect defects in use case documents, which proved effective when evaluated on five different documents. Tianual and Pohthong (2019) proposed a model-based validation technique using form-based specification and UML use case description to detect defects during analysis or requirements engineering phases. Their evaluation showed better defect detection than manual methods but with increased time consumption. Törner et al. (2006) reported on the quality assessment of use cases at Volvo Car Corporation and found that the defect types with the highest defect intensity were missing element and incorrect linguistics.

The findings suggest that the success of use cases to reduce requirements defects depends on their being defect-free and the appropriate implementation of the behaviour they communicate through development practices. This indicates that use cases do not directly identify defects as they require other techniques to ensure they are defect-free. However, defect-free use cases can help reduce misunderstandings of requirements which can eventually prevent occurrence of defects during development. Therefore, the use of use case is recommended as an option for describing and organising system requirements. This can ultimately help to prevent defects in MDS development by providing a clear and structured representation of the system's behaviour. However, when using use cases developers must ensure that they accurately capture and present the correct behaviour of the system to avoid introducing defects.

Findings on Product Backlog

Out of 667 studies analysed, 14 were on product backlog and none of these discussed using it to address software defects. Most of these studies discussed improving defect management using the product backlog. For example, Staron et al. (2010) presented a method for forecasting the level of defect backlog in large, lean-based software projects that proved better at predicting the level of defect backlog than the absolute number of defects. Ahamed (2007) proposed two metrics, Backlog Management Index (BMI) and Backlog Clearance Planning Index (BCPI), for effective defect backlog management. Studies such as Kokkola (2004), Doherty et al. (2016), and Sedano et al. (2019) suggest that the product backlog is used to manage defects by adding them to the product backlog and prioritising for resolution in future iterations. These findings indicate that while the product backlog may not directly help in detecting defects, it

serves as a valuable management tool for organising items for future iterations or sprints. However, it is not recommended as a means of identifying and removing defects in MDS development.

Findings on Sprint Backlog

Academic studies on sprint backlog are scarce. Out of 667 studies analysed, six were on sprint backlog and none of these discussed it in relation to software defects. To date no research has been found that discusses sprint backlog in relation to software defects, indicating that its main purpose is to manage items for the sprint rather than addressing defects. Therefore, sprint backlog is not recommended for addressing defects in MDS development.

Findings on Sprint

To date no research has been found suggesting that sprint can be used to address defects. Out of 667 studies analysed, six were on sprint and none of these discussed it in relation to software defects. Instead, the six studies focused on other aspects such as factors affecting sprint effort estimation (Ramessur and Nagowah 2020), sprint performance forecasts in ASD (Kortum et al. 2019), and optimal sprint length determination (Anand et al. 2021). The findings suggest that sprint does not detect and remove defects. Therefore, it is not recommended for addressing defects in MDS development.

Findings on Sprint Planning Meeting

Out of 667 studies analysed, three were on sprint planning meeting and none of these discussed it in relation to software defects. These studies discussed other aspects such as role of sprint planning and feedback in game development projects (Liu et al. 2019) and sprint planning optimisation in data warehouse design (Golfarelli et al. 2012). The dearth of academic literature on sprint planning in relation to software defects suggests that its primary purpose is planning the sprint rather than detecting and removing defects. Therefore, it is not recommended as a means of addressing defects in MDS development.

Findings on Sprint Review Meeting

Academic literature on sprint review meeting is scarce. Out of 667 studies analysed, only three were on sprint review meeting. While Fan (2018) mentioned the aspect of corrective feedback in connection to sprint review meeting, none of the three studies discussed the sprint review meeting in relation to addressing software defects. This suggests that sprint review meetings

are not used to detect defects in software. Therefore, it is not recommended for addressing defects in MDS development.

Findings on Model Storming

Academic studies on model storming are scarce. Out of the 667 studies analysed, three mentioned model storming but did not specifically focus on this practice alone. These studies covered Agile Model Driven Development (AMDD) and mentioned model storming as a core practice within this approach. Despite the potential for model storming to address defects as indicated in Ambler (2007), to date no study has been found discussing its application for this purpose. This indicates that research on model storming in relation to defects has not been conducted or published. Considering its functionality, model storming can be useful in preventing defects in MDS development by developing a clear model of a system or its components. This can reduce misunderstandings of requirements by developers and prevent defects from arising. Broadly, models offer an effective way to investigate, revise and communicate system aspects to stakeholders. Therefore, model storming is recommended as a means of addressing misunderstandings of requirements in MDS development. Model storming can help to clarify requirements and prevent defects by facilitating collaboration and communication among developers and stakeholders. However, as with any practice, it is important for developers to have a clear understanding of how model storming works before using it. The lack of understanding may result in the creation of incorrect models and the introduction of defects.

Findings on Planning Game

Academic literature on planning game is scarce. No studies were found that specifically focused on planning game alone. Instead, this practice is commonly mentioned in literature on XP and was analysed in nine studies on XP. None of these studies discussed planning game in relation to addressing software defects. Based on the lack of evidence, planning game is not recommended for addressing defects in MDS development.

4.3 Discussion

The analysis has revealed that agile practices such as PP, TDD, CI, and onsite customer address software defects in GSD. However, research exploring the use of agile practices for addressing defects in MDS development remains limited. The findings show that among the analysed agile practices, only onsite customer has been specifically reported for addressing defects in an MDS

development project. To bridge this gap, further research is necessary to gather first-hand insights from MDS organisations regarding the use of agile practices for addressing defects and how they are applied. Based on the review and analysis performed in section 4.2, 17 of the 25 agile practices are recommended for addressing defects in MDS development. These include PP, TDD, CI, unit testing, refactoring, integration testing, onsite customer, code review, UAT, code ownership, coding standards, small releases, user story, simple design, use case, BDD, and model storming. Additionally, practices such as refactoring, unit testing, integration testing, UAT, onsite customer, and PP can also identify defects in the later stages of MDS development. The steps used to identify the 17 agile practices are outlined below.

1. Identify and review agile methodologies. The review of various agile methodologies and their practices is provided in section 2.7.2.
2. Gather agile practices from the reviewed agile methodologies. In total 84 agile practices were gathered. This is detailed in section 2.10.
3. Perform inclusion and exclusion of the 84 agile practices, leading to 25 agile practices. This is detailed in section 2.10. The process included the following steps:
    i. Examine the practice based on the functional description in relation to addressing defects
    ii. Examine the practices based on similarity in naming and functional description – practices with similar names and functional descriptions, one was returned the other was excluded.
    iii. Assess the potential impact a practice could have at the stages of development, particularly in relation to addressing defects.
4. Perform further evaluation of the 25 agile practices to validate their inclusion, specifically focusing on addressing defects. This is detailed in in section 2.11.
5. Perform review and analysis of the 25 agile practices to evaluate the evidence of their use for addressing defects in the GSD and to determine whether, and how they could be applied to MDS development for addressing defects. This process led to selection of the 17 agile practices integrated in the AV4AD model.

Based on the findings from the analysis, this research advocates that some of the recommended agile practices can be used in combination to achieve even more better results in addressing

defects. The proposed combinations are shown in Table 4.8. The agile practice in the left column in Table 4.8 can be used in combination with any of the practices in the right column.

Table 4.8: Suggested practices that can be used in combination

Practice	**Can be used in combination with**
PP	TDD, onsite customer, UAT, user story, simple design, coding standards, code ownership, unit testing, integration testing, use case
TDD	onsite customer, user story, UAT, refactoring, simple design, coding standards, code ownership, BDD, use case
Refactoring	PP, unit testing, integration testing, UAT, onsite customer, coding standards, code ownership
Unit testing	PP, UAT, onsite customer
Simple design	PP, onsite customer, UAT, model storming
Small releases	PP, TDD, UAT, onsite customer, refactoring
Use case	PP, TDD, UAT, onsite customer, BDD, model storming
Integration testing	PP, onsite customer, UAT, refactoring, unit testing
User story	PP, TDD, onsite customer, model storming, BDD

From the findings of the analysis, the following agile practices have not been recommended for addressing defects in MDS: sprint, planning game, sprint review meeting, sprint planning meeting, sprint backlog, product backlog, daily stand-up meeting, and retrospective.

4.4 Agile Practices used for Addressing Defects in GSD and MDS domains

As shown in table 4.9 column 3, a high number of studies used agile practices such as sprint, sprint planning meeting, sprint backlog, sprint review meeting, and retrospective in MDS development, but none discussed using them specifically to address software defects as shown in column 4. Similarly, planning game and use case were not used for addressing defects. This may indicate a lack of research or direct impact on coding and testing during software development.

Table 4.9: Articles reporting agile practices for addressing defects in GSD and MDS

Agile practice	Defects in GSD	MDS	Defects in MDS
Test-Driven Development	38	13	0
Unit Testing	25	16	0
Refactoring	23	10	0
Code Review	19	2	0
Pair Programming	18	12	0
Continuous Integration	10	17	0
Code Ownership	5	3	0
Coding Standards	5	5	0
Small Releases	5	1	0
Behaviour Driven Development	4	0	0
Onsite Customer	3	6	1
User Acceptance Testing	3	5	0
Integration Testing	2	4	0
User Story	1	17	0
Simple Design	1	4	0
Product Backlog	0	17	0
Daily Stand-up Meeting	0	15	0
Planning Game	0	2	0
Use Case	0	6	0
Sprint/Iteration	0	20	0
Sprint Planning Meeting	0	15	0
Sprint Backlog	0	11	0
Sprint Review Meeting	0	8	0
Retrospective	0	13	0
Model Storming	0	0	0

Several studies used TDD, PP, unit testing, refactoring, and CI for MDS development and to address defects in GSD. The high number of studies using these practices in GSD to address defects may indicate their direct impact on design, coding, and testing during software development. A total of 29 studies discussed various practices in relation to addressing defects in GSD: one study focused on user story, five on coding standards, three on onsite customer,

five on code ownership, two on integration testing, one on simple design, three on UAT, five on small releases, and four on BDD. One study reported significant improvement in software defect identification early in the development process of MDS because of onsite customer/user involvement (Tang et al. 2018). No study discussed using model storming to address defects despite its potential to resolve requirements defects collaboratively and quickly through creation of models (Ambler 2007; Alshazly et al. 2014). Although few studies used code review in the MDS domain, it was used in several studies in GSD to address defects.

4.5 Survey of Medical Device Software Organisations

To complement findings of the literature review in Chapter 2 and the analysis in section 4.2, it was deemed important to gather insights into which agile practices are currently used in MDS development and whether they are used specifically to address defects. A survey was chosen as the most effective approach for this purpose. The insights gained through this survey were important in the development of the AV4AD model, which is useful for any client wishing to address defects during MDS development. The details of the survey process are provided in section 3.9. The survey instrument and accompanying materials that were used in this research can be found in Appendix C.

4.5.1 Survey Results Analysis

Fifteen responses were received from different MDS organisations, including twelve SMEs and three large organisations, as per the European Union's definition of Small and Medium-sized Enterprises (SMEs) (EU 2022). The survey results revealed varied experience in MDS development among the organisations ranging from 1 year to over 20 years. The results also revealed that 60% of organisations, including 7 SMEs and 2 large organisations, use exclusively agile methodologies for MDS development. A small number, 7%, of SMEs use exclusively the V-Model, while another 7% of SMEs use both the V-Model and Waterfall. Lastly, 27% of the organisations, comprising 3 SMEs and 1 large organisation, use a combination of both plan driven and agile approaches.

Of the 60% that use exclusively agile methodologies, 43% SMEs have used them for about 5 years and the remainder for about 15 years. One large organisation has used an agile methodology for less than 5 years, and the other for less than 15 years. The 7% SMEs that use exclusively the V-Model, have used it for about 15 years. The other 7% of SMEs that use both the V-Model and Waterfall have used these for about 20 years. Of the 27% using a hybrid

methodology, 33% SMEs have used it for over 20 years and the remainder for less than 5 years. The remaining large organisation has used the hybrid approach for about 20 years.

The reasons for organisations using their methodology were accommodating changing requirements (80%), addressing defects during development (67%), reducing time to market (60%), and both reducing development costs and meeting regulatory requirements (47%) each. However, two-thirds of the large organisations and a quarter of SMEs did not indicate addressing defects as one of the key aspects in their methodology implementation.

The survey also revealed that most organisations address defects during Testing (93%), Coding (87%), Release (80%), and Maintenance (73%). It also revealed that 87% of the organisations use the agile practices shown in Table 4.10 for addressing defects, with Code Review, Unit Testing, Integration Testing, and Coding Standards as the commonly used. Additionally, 85% of the organisations that use agile practices for addressing defects implement them commonly during coding and testing stages as shown in Table 4.11.

The survey revealed 60% of the organisations use exclusively an agile methodology. This indicates a significant increase from the previous survey that reported 25% of organisations were using agile methodologies for MDS development (McHugh, McCaffery and Casey 2012). The shift towards agile practices can be attributed to the research carried out over the years that identified and addressed the challenges of using agile practices for MDS development (McHugh, McCaffery and Casey 2012).

The survey results also reveal that when addressing defects in MDS development, emphasis is on defect identification. As shown in Table 4.11, most organisations carry out this process typically during the coding and testing stages of the development lifecycle. This is also corroborated by a high number of organisations that indicated addressing defects during Testing (93%), Coding (87%), Release (80%), and Maintenance (73%). This finding aligns with what is suggested in the literature that in many software development projects, defect prevention is often overlooked or inadequately implemented using inefficient methods (Jamal et al. 2015). Prioritising defect identification while neglecting defect prevention can lead to costly consequences (Huang and Liu 2017).

Although there is similarity in the agile practices identified from the literature and those from the survey (Table 4.9), an observable difference exists in the agile practices commonly used for addressing defects. The literature review revealed that PP, TDD, Code Review, Unit

Testing, and Refactoring are the commonly used practices for addressing defects in the non-MDS domain. However, the survey revealed that Code Review, Unit Testing, Integration Testing, and Coding Standards are the commonly used practices in MDS development. This affirms the focus on defect identification in MDS development as these practices are typically applied after actual coding is complete - when all the mistakes have been made and resources wasted. However, Coding Standards is an exception as it is applied during actual coding. While these practices are effective in identifying defects, in MDS development, emphasising or solely relying on defect identification can lead to costly rework and a higher risk of the final MDS failure.

Table 4.10**:**. Agile practices usage for addressing defects in MDS development

Agile practice	**Use for addressing defects**
Code Review	85%
Unit Testing	77%
Integration Testing	77%
Coding Standards	69%
User Story	62%
User Acceptance Testing	54%
Refactoring	54%
Continuous Integration	54%
Test-Driven Development	46%
Small Releases	38%
Pair Programming	38%
Use Case	31%
Code Ownership	23%
Onsite Customer	23%
Behaviour-Driven Development	15%

Table 4.11**:**. Stages where agile practices are used for addressing defects in MDS development

Software development stage	**Agile practices usage**
Requirements	38%
System Design	38%
Architectural Design	38%
Module Design	46%
Coding	85%
Testing	85%
Release	38%
Maintenance	46%

4.5.2 Discussion

The survey findings show that despite a higher number of organisations using agile methodologies than traditional ones, MDS is developed using both approaches. Therefore, when planning a solution for addressing defects during the development lifecycle of MDS, it should accommodate both categories of organisations (those using plan-driven approaches and those using agile methodologies). Additionally, the findings show that MDS organisations typically focus on defect identification. Consequently, the solution for addressing defects should emphasise defect prevention to ensure defects are detected and fixed early, while also incorporating defect identification. As discussed in previous chapters, defects that slip to the later stages of the software development lifecycle can result in costly rework and even software project failure. Moreover, defects that arise during use can lead to catastrophic consequences for both patients and manufacturers.

4.5.3 Survey Limitations

There are several potential limitations associated with the survey conducted in this research. Firstly, the low number of responses received makes it difficult to generalise findings to the entire population of MDS organisations. As with any survey, it was practically impossible to receive responses from all the 65 organisations. However, as mentioned in section 3.9, the primary focus in this study was to gather first-hand insights into MDS development and the use of agile practices for addressing defects. Therefore, the collected data was relevant and sufficient to address the objectives of this research. Secondly, the low response rate may have

introduced a degree of bias as non-respondents' views could differ from those who responded. Nevertheless, the responses were received from both SMEs and large organisations, reducing the bias in the collected data. Moreover, the insights gained through the survey provided valuable information on which agile practices are currently used and how they are used for addressing defects within MDS development organisations.

4.6 Approach to the Development of the AV4AD Model

As discussed in section 2.3, defects in MDS can lead to costly rework and even software failure, causing catastrophic consequences. Findings from the literature review and survey indicate that MDS organisations have not adequately leveraged agile practices to address defects despite their wide use in the GSD for this purpose. Moreover, the findings indicate that an exclusive emphasis on defect identification is insufficient for MDS development. It is important in MDS development to adopt a holistic approach that prioritises defect prevention and is complemented by defect identification. The insights gathered from both the literature review, survey, and analysis were instrumental in devising a solution for addressing defects throughout the MDS development lifecycle.

The literature review revealed that among the various software development approaches, the V-Model is the most commonly used for MDS as it produces deliverables required for regulatory compliance. Therefore, the V-Model was chosen as the foundation for the proposed solution. Having decided to base the solution on the V-Model, the research proceeded to the development of the AV4AD model that is the core contribution of this research. The AV4AD model was developed by integrating agile practices identified in section 4.2 that address defects with the V-Model. The AV4AD model is designed to assist MDS development organisations to prevent defects and identify those that may slip through to the later stages of the development lifecycle. It is designed for use by any organisation developing MDS, irrespective of organisational size or MDS classification. The following subsections provide further details on the choice of the V-Model, followed by the structure of the AV4AD model.

4.6.1 Selecting the SDLC Model for Agile Practices Integration

Based on the literature review detailed in Chapter 2, as well as the review, analysis, and survey presented in this chapter, it was determined that a solution should be developed to assist both organisations that use traditional approaches and those that use agile methodologies. This solution supports values of both approaches, namely people and process, which is essential for

MDS development. It was also determined to align the solution with the IEC62304 standard which defines the software development lifecycle processes for MDS. To support both traditional and agile organisations effectively, a hybrid approach that specifically focuses on addressing defects throughout the development lifecycle of MDS was deemed most suitable. Therefore, it was necessary to identify which plan-driven approach could be the best foundation into which agile practices could be integrated. The exploratory literature review on MDS development revealed that the V-Model is the most commonly used traditional approach in MDS development. This finding is corroborated with the literature review detailed in section 2.5. The V-Model's dominance is attributed to various factors, including its capability to produce deliverables required for regulatory compliance. Regulatory institutions mandate MDS organisation to produce deliverables related to verification and validation activities. These activities are inherently supported in the V-Model, making it particularly appropriate for MDS development. Although the survey results indicated that both the waterfall and V-Model are used, the V-Model appeared to be the most prevalent.

Based on the literature review and survey findings, the V-Model was chosen as the foundation for the development of the AV4AD model. Agile practices that address defects were identified through the literature review, analysis and survey of MDS organisations. From the 25 agile practices, 17 have been integrated into the V-Model, forming the AV4AD model. The 17 agile practices were selected based on an analysis of their effectiveness in addressing defects in the GSD, their documented use in MDS development as outlined in Chapter 2, and their potential to address defects within MDS projects. These practices include PP, TDD, CI, unit testing, refactoring, integration testing, onsite customer, code review, UAT, code ownership, coding standards, small releases, user story, simple design, use case, BDD, and model storming. The AV4AD model was initially reviewed by six software professionals within the Regulated Software Research Centre at DkIT. The feedback obtained from the reviews helped refine the model. The AV4AD model is shown in Figure 4.1.

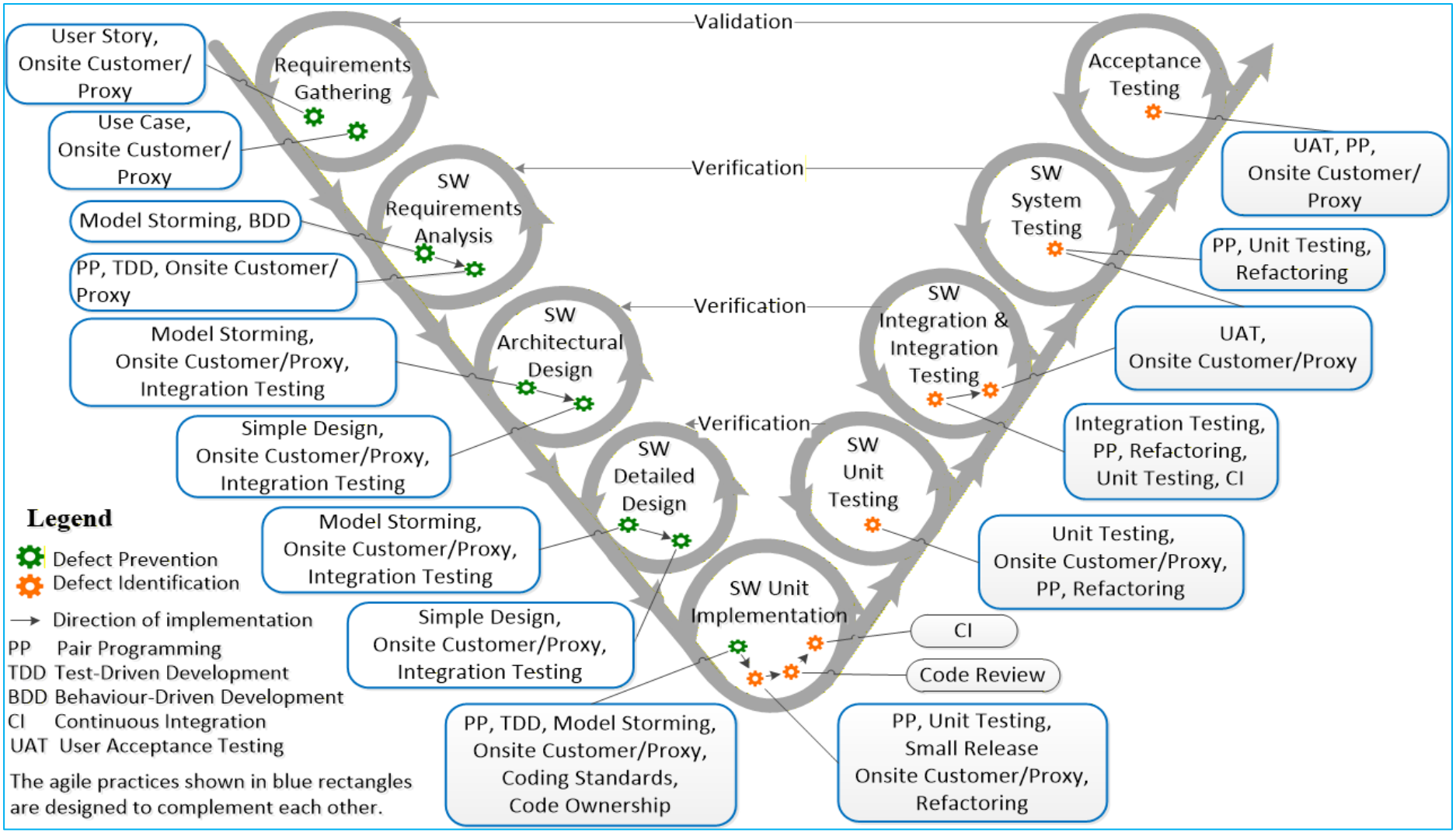


Figure 4.1: The AV4AD Model

4.6.2 Structure of the AV4AD Model

The model aligns with the IEC62304 – Medical Device Software – Software life-cycle processes (AAMI/IEC 2015) as it adopts the lifecycle processes of this standard. The AV4AD model consists of nine stages with verification and validation activities that are inherent in the V-Model. As shown in figure 4.1, agile practices applied in the early development stages are used for preventing defects, while those applied in the later development stages are used for identifying defects. The model emphasises involvement of the customer or proxy throughout the development lifecycle. Customer involvement in the early development stages helps prevent requirements defects and design defects early on (Alvertis et al. 2016). This reduces unnecessary costs that would be incurred when these defects arise in later stages.

Agile practices in the AV4AD model are designed to complement each other. These practices can be used together or individually in a stage in which they are included. This provides MDS developers the flexibility to choose a single practice or a combination of practices from a particular stage and use to address defects at that stage. For example, during the requirements gathering stage, appropriate agile practices can be selected as required. Additionally, where multiple or all agile practices are selected for implementation at a particular stage, the direction of implementation is recommended as a best practice to enhance defect prevention or identification. For example, at the software requirements analysis stage, it is recommended to start with Model Storming and BDD followed by PP and TDD, and finally Onsite Customer/Proxy.

Unlike the V-Model, the AV4AD model encompasses a continuous and iterative process at each stage and provides the flexibility to refer to any previous stage depending on the magnitude of the detected defect. This ensures that defects detected at subsequent stages, which require fixing at any prior stages, are addressed before proceeding to the next stage. For instance, if a defect is detected during the software detailed design stage and requires fixing at the software architectural design stage, it will be fixed at the software architectural design stage before proceeding to the implementation stage. Similarly, if a defect is identified at the implementation stage and requires significant modifications at the requirements analysis stage, it will be fixed at the requirements analysis stage before progressing to the software integration and integration testing stage.

At the early stages of many software development projects, clients are often not clear about their system requirements (Haleem et al. 2021). Research has shown that many defects in software development can be traced back to the early stages, especially during the requirements elicitation, requirements analysis, and design stages (Suma and Nair 2010). Therefore, applying agile practices that foster interaction between clients, programmers, system architects, and testers during the early stages can assist to prevent defects such as requirements defects (Michael and Boniface 2014). Defects that originate from the early stages like requirements analysis stage, can have a devastating impact throughout the development lifecycle of MDS. These defects significantly contribute to system failures, particularly when they arise in later stages (Kamalrudin et al. 2018). The following sections discuss the implementation of agile practices in the AV4AD model.

4.6.3 Requirements Gathering Stage

At this stage, the AV4AD recommends the application of a combination of either Onsite Customer and User Story or Onsite Customer and Use Case. At this stage, requirements should be collected and presented in form of user stories or use cases that are clear and easy to comprehend. These should be written or developed in close collaboration with customers who can clarify any misinterpretations and oversights early, preventing requirements defects in the succeeding stage. Therefore, active communication among stakeholders is a key element at this stage and should be nurtured throughout the requirements gathering process for MDS. This ensures that the user stories or use cases represent what the customer wants about the system. Well-written or developed user stories or use cases can minimise the likelihood of defects in the later stages of MDS development. Ensuring thorough review and refinement of a user story or use case at this stage is essential for preventing requirements defects that can lead to costly consequences in later stages.

4.6.4 Software Requirements Analysis Stage

The primary goal of using agile practices at this stage is to prevent various types of requirements defects such as ambiguities and omissions (Alshazly et al. 2014; Margarido et al. 2011). This ensures that such defects do not arise in later development stages where they become costly to find and fix (Basak and Shazzad Hosain 2014). Traditionally, TDD and PP are applied during the coding activity. While these practices are useful for identifying defects during this activity, applying them in the early stages of MDS development can be more beneficial in preventing defects. Moreover, restricting their use to the coding activity may lead

to a significant amount of time and other resources being spent on defects that would otherwise be addressed during the early stages. The AV4AD model recommends application of a combination of Model Storming and BDD followed by a combination of TDD, PP, and Onsite Customer at this stage. This approach ensures that software requirements defects are caught and prevented early. The key element that makes this approach practical at this stage is open and frequent communication among clients, programmers, system architects, and testers. The conversations among stakeholders involved in the Model Storming and BDD process and the demonstration of real examples, ensures that requirements defects are caught and prevented early. Similarly, the conversations that occur when pair programmers or testers alternate writing and reviewing test scenarios and test cases for requirements, and demonstrating functionalities to customers and receiving feedback, ensures that potential requirements defects are prevented early on at this stage. These practices should be applied iteratively for each user story or use case.

4.6.5 Software Architectural Design Stage

An important output of this stage is the software architecture that shapes the overall vision of the software product (Dasanayake et al. 2019). A well-designed software architecture is the foundation to a successful software product as it serves as a blueprint for developers. It enables software engineers to understand the structure of a software system, its components, and the interrelations among these components (Kouroshfar et al. 2015). As discussed earlier, any defect originating from this stage if not detected and resolved at this stage, can propagate to the subsequent stages and lead to adverse consequences and ultimately final software product failure. At this stage, the AV4AD model recommends an iterative and complementary application of Model Storming, Onsite Customer/Proxy, and Simple Design to prevent architectural design defects in MDS. This approach operates on the principle of having architects develop and demonstrate architectural design models to customers and obtain feedback. Regular customer reviews ensure that architectural decisions align with business needs, ultimately preventing defects.

When an understanding is reached between architects and all stakeholders including customers, the software architecture is developed and refined to achieve a simple design to allow easy comprehension and actual coding by the programmers during the coding activity. This ultimately allows speedy and timely delivery of the software product to customers as less time is spent on defects during coding (Wan et al. 2023). Additionally, the AV4AD model

recommends that Integration Testing should be an integral part of the entire architectural design process to resolve potential integration defects early. Essentially Integration Testing at this and the next stage is used to mean involvement of integration testers. Involving integration testers at this stage helps them to gain a clearer understanding of the software components and their interactions and developing integration test cases early for verifying the components during the software integration stage. Therefore, the important element of integration testing at this stage is that potential integration defects are caught and prevented early as integration testers can detect these defects when considering the interrelations among the components.

4.6.6 Software Detailed Design Stage

The key output at this stage is a well-defined software detailed design that specifies the logical structure of the components and their functionalities, showing interfaces to interact with other modules. Defects stemming from this stage can lead to adverse consequences if they occur during the later stages. To ensure that defects are prevented early at this stage, the AV4AD model recommends an iterative and complementary application of Model Storming, Onsite Customer, Integration Testing, and Simple Design. Involving customers at this stage allows their input on the models to be incorporated early enough, thereby influencing the design of the final MDS product that aligns with their business needs. Similar to the preceding stage, Integration Testing should be an integral part of this stage to catch potential integration defects early. Integration testing at this stage involves assessing the overall structure and behaviour of the software system to catch potential design and integration defects, as well as creating integration tests for verification during integration and integration testing stage. The structured application of agile practices at this stage can prevent design and integration defects from propagating to later stages of MDS development and ensure that a simple design is achieved. A simple design helps programmers to easily comprehend module functionality and write efficient code – code that accurately accomplishes the functionality stated in the requirement.

4.6.7 Software Unit Implementation Stage

The desirable output at this stage is defect-free and efficient code. To achieve this, a clear understanding of the requirements is essential at this stage. This is primarily aided by the thorough defect prevention processes carried out in the preceding stages. Nonetheless, defect prevention can also be carried out at the outset of this stage and during the actual coding. The AV4AD model recommends applying a combination of Model Storming, PP, TDD, Onsite

Customer, Coding Standards, and Code Ownership to prevent defects at this stage. Once code for a functionality is written, defect identification begins, starting with a combined application of PP, Unit Testing, Small Release, Onsite Customer, and Refactoring. This is then followed by application of Code Review, looping back to the first activity when code has not met the set review standard. Code that has met the review standard is added to the codebase through Continuous Integration process in which the build and test activities can further identify defects. This systematic application of agile practices at this stage ensures that defects are prevented in the first place, and those that escape this activity are identified and fixed before the next stage.

4.6.8 Software Unit Testing Stage

At this stage, the AV4AD model recommends the application of Unit Testing, Onsite Customer, PP, and Refactoring. Onsite Customer should follow Unit Testing to verify desired functionality. Defects identified through Unit Testing and Onsite Customer are assessed by the pair programmers. Minor defects are resolved at this stage through refactoring while larger defects requiring fixing at any specific preceding stage are referred and fixed there. For minor defects, these agile practices should be iteratively applied to ensure changes made during refactoring do not introduce any defect. Thus, implementation of Refactoring and PP followed by Unit Testing at this stage not only resolves the identified minor defects but also avoids introducing new ones. This systematic implementation of these agile practices at this stage reduces the risk of defects escaping into the subsequent stages. While agile practices included at this stage might be used, in implementing the AV4AD model, many defects would be prevented in the early stages. By proactively applying agile practices at earlier stages, defects are minimised before they reach later stages. As a result, fewer defects are expected to slip through to the software unit testing stage.

4.6.9 Software Integration and Testing Stage

Considering that at this stage individual modules or subsystems are combined and work together as a single system, defects that escape this stage can cause the final MDS to fail. At this stage, the AV4AD model recommends using a combination of Integration Testing, PP, Refactoring, and Unit Testing followed by Onsite Customer and UAT to review and verify the integrated components or system. This approach ensures that defective modules or subsystems are identified, refactored and retested at this stage depending on the magnitude of the defect.

The retested modules are reintegrated through Continuous Integration where the build and test activities can further identify defects. Demonstrating the integrated system to customers ensures that defects arising due to deviations from desired reliability and functionality are identified and corrected. This systematic approach increases the likelihood of identifying potential defects as defective modules can be refactored and retested at this stage. This minimises the risk of defects escaping to the subsequent stage.

When a defect identified during this stage requires correction in earlier stages, the involvement of programmers and testers at this stage facilitates a thorough understanding of the scope of necessary modifications to address it. Additionally, testers can assess whether new tests must be written for retesting the corrected component. However, as in the preceding stage, the implementation of the AV4AD model is expected to result in fewer defects at this stage when agile practices are proactively applied during the early stages. Therefore, agile practices included at this stage might be minimally used.

4.6.10 Software System Testing Stage

At this stage, the AV4AD model recommends using UAT, PP, Onsite Customer, Refactoring, and Unit Testing. Any defect identified by customers through acceptance testing are evaluated by the pair programmers and testers. Defects considered minor enough are fixed immediately at this stage by performing minor refactoring and unit testing. Defects requiring to be fixed at the preceding stage are referred to that stage, while those requiring to be fixed at much earlier stages are referred and resolved at those stages. Applying Refactoring and PP followed by Unit Testing not only resolves the identified minor defects but also avoids introducing new ones. A structured implementation of these practices at this stage ensures that defects that could potentially escape to the release of the MDS are caught and resolved. Although agile practices included at this stage might be used, in implementing the AV4AD model, many defects would be prevented in the early stages. By proactively applying agile practices at earlier stages, defects are minimised before they can reach the software system testing stage. Consequently, fewer defects are expected during this stage, thereby reducing rework costs.

4.6.11 Acceptance Testing Stage

At this stage, the AV4AD model recommends applying UAT, Onsite Customer, and PP. Defects identified through UAT and Onsite Customer are evaluated by pair programmers. Minor defects are referred to the preceding stage and are fixed there. Defects considered to be

major are referred and fixed at the specific prior stage. The systematic implementation of these practices at this stage ensures that issues, including functional and interface defects are caught and fixed before the final release and use of MDS. Defects that escape to the final release of MDS can lead to devastating consequences. Similar to the above three stages, the implementation of the AV4AD model is expected to result in fewer or no defects at this stage when agile practices are proactively applied during earlier stages. Consequently, agile practices included at this stage might be minimally applied.

4.7 Summary

This chapter presented the approach to the development of the AV4AD model. It detailed the review and analysis performed to identify suitable agile practices that can address defects in MDS development. A total of 25 agile practices were reviewed and analysed resulting in the identification of 17 suitable practices. The chapter also discussed the survey of MDS organisations conducted to gather first-hand insights into MDS development and whether agile practices were used for addressing defects and their application.

The review and analysis revealed that agile practices are used for addressing defects in the GSD. However, MDS organisations have not adequately leveraged agile practices to address defects. The survey revealed a strong emphasis on defect identification, particularly during coding and testing stages. The commonly used practices during these stages are Code Review, Unit Testing, Integration Testing, and Coding Standards. The emphasis on defect identification is consistent with findings from the literature review which revealed that MDS organisations rely on software testing after development is completed. In MDS development, it is important to adopt an approach that prioritises defect prevention and is complemented by defect identification. Based on insights from the literature review and survey, the AV4AD model was developed and presented in this chapter. The AV4AD model has been developed to assist MDS organisations to address defects throughout the MDS development lifecycle. The key distinction of the AV4AD model, compared to existing models, is that it enables MDS organisations to address defects at each stage throughout the MDS development lifecycle. This is achieved by agile practices that prevent defects in the early stages and identify defects in the later stages. This approach significantly reduces the risk of defects slipping to the final software release, ultimately avoiding costly consequences for MDS organisations and preventing patient injuries and deaths.

5 Evaluation and Validation of the AV4AD Model

5.1 Introduction

This chapter presents the evaluation and validation of the AV4AD model as part of the DSRM's Activity 4 (Demonstration), and Activity 5 (Evaluation) as outlined in Chapter 3. The evaluation addresses part of *RO5*: *To develop and evaluate a model that incorporates agile practices to address defects throughout the MDS development lifecycle*. The validation addresses *RO6: To validate the model*. The AV4AD model was evaluated through semi-structured interviews conducted with MDS developers from 8 MDS organisations in Ireland. This was then followed by validation within two MDS organisations in Ireland. A detailed account of the semi-structured interview process is presented in section 3.10. The findings from both the evaluation and validation approaches are reported in this chapter.

5.2 Evaluation

5.2.1 Analysis of the Semi-structured Interviews

A total of 155 pages of transcription text was generated from the semi-structured interviews. Due to the semi-structured nature of these interviews, the transcriptions consisted of both relevant data and data that is not related to the research. Therefore, a systematic approach was required to extract pertinent insights to the research. To achieve this, the five stages for analysing semi-structured interviews specified by Schmidt et al. (2004) were followed. These stages include the following: 1) material-oriented formation of analytical categories, 2) assembly of the analytical categories into a guide for coding, 3) coding of the material, 4) quantifying surveys of material, and 5) detailed case interpretations.

Stage 1 involved the researcher's familiarisation with the collected data through repeated viewing and listening to the video recordings of the interviews, as well as repeated reading of the transcripts. Through this process analytical categories aligned with interview questions were identified and distinguished using different colours to facilitate coding. For example, bright green was used to indicate affirmation of practices and the AV4AD model for addressing defects, whereas light blue indicated discussions related to standards. Additionally, notes were included for each analytical category.

Stage 2 entailed reviewing and sorting these analytical categories and forming a guide for coding. Additionally, codes were developed and their relationships examined. Stage 3 involved

actual coding of the transcription text and identifying themes and sub-themes. During this stage, the researcher worked through each transcript highlighting and coding segments of the text that appeared to be relevant to the research. A code book, containing a list of codes, descriptions, and references to corresponding categories in the transcripts is provided in Appendix F. Stage 4 focused on examining the relationships between the identified themes and sub-themes and reviewing and finalising the codes. Finally, in stage 5, a detailed interpretation of the data was performed.

The following is the list of main themes that were covered in the semi-structured interviews. The individual feedback from interviewees is presented next.

- Agile practices included in the AV4AD model
- Complementary practices
- Defect prevention
- Defect identification
- AV4AD model implementation and conformance to IEC62304
- AV4AD model comparison to organisation's current approach in addressing defects
- Adoption of the AV4AD model in participant's organisation
- Practice(s) that the participant's organisation would promptly adopt
- Improvements to the AV4AD model
- Recommending the AV4AD model in MDS development

5.2.2 Interviewees Feedback

Interviewees provided feedback on the efficacy and suitability of the AV4AD model to address defects and its potential improvements. The feedback was collected through discussions initiated by the interview guide questions and probes generated from their responses. This section presents the interviewees' feedback on the themes presented in section 5.2.1. To ensure familiarity with the model and allow quality feedback, all interviewees were provided with the AV4AD model and a document describing it through email prior to the interview. During the interview sessions, the AV4AD model was also shared on the screen for reference as they provided their feedback. At the start of each interview, interviewees were asked if they required further explanation about the model or were ready to proceed with answering questions and discussing the model.

5.2.2.1 Interviewee 1 Feedback

During the interview, interviewee 1 examined the model focusing on the agile practices at each stage of the AV4AD model and stated that they are suitable for addressing defects. They mentioned that in their organisation's current development approach agile practices are specifically used during coding and testing stages. This may indicate a focus on defect identification. Interviewee 1 discussed how their organisation would benefit from using the AV4AD model to address defects. Starting with the left side stages and the coding stage of the model, they stated that agile practices in these stages would prevent defects. They further examined the complementary use of practices in these stages and affirmed their potential to enhance defect prevention. Specifically, when discussing practices at the software architectural design stage, interviewee 1 affirmed that all practices would contribute to preventing defects, stating: "*At this point, they make sense together. Some type of high level design there, makes sense. And then again integration testing and onsite customer makes sense there.*". However, they expressed uncertainty about Model Storming. They stated:

> "*So, the only one in this section I wouldn't be sure of is just the model storming. I am not sure like at that point should you be just working off the requirements. Certainly, could help reinforce. Yeah, my instinct would be at that point to leave the model behind, but yeah, I will consider this approach*."

Their uncertainty about Model Storming at this stage may stem from their organisation's current plan-driven approach which contrasts with the 'just in time' nature of Model Storming which would enhance defect prevention. Despite their initial uncertainty, Interviewee 1 said that they would use Model Storming throughout all the left stages of the model including the coding stage. They stated: "*Model storming all the way down, we are probably gonna be using this or something close to it in our next production project. My instinct is it probably will improve the quality of requirements*".

In discussing agile practices at the software unit implementation stage, interviewee 1 emphasised their importance for addressing defects at this stage and their proposed complementary application. They affirmed, "*Yes, they make sense. Yes, they all make sense together*". However, they expressed uncertainty about using Model Storming during this stage in reference to its current limited application during requirements analysis stage in their organisation. Despite their uncertainty, they asserted its usefulness for addressing defects at

this stage, stating, "*But again, I totally see the value of it as a tool to constantly reinforce the problem you are trying to solve with the software*".

Examining agile practices for addressing defects at the software unit testing stage and right side stages of the AV4AD model, interviewee 1 stated, "*Yes. So, they are very useful in identification, whether manual or automated*". Additionally, they affirmed the complementary use of practices in these stages, stating, "*Yes, they are all complementary for defects identification*". They specifically recommended the complementary use of PP and unit testing or PP and integration testing at this stage and the software system testing stage. They suggested pairing two testers, a developer and a tester or even forming multiple pairs of testers. They stated:

> "*The pair programming, in this case would typically be a tester and another tester. So, sometimes a tester working with a developer, which can be helpful…. But it would be useful if it was pair programming with multiple testers, would be even better, you know*".

However, interviewee 1 expressed uncertainty about the role of unit testing at the software integration and integration testing stage as well as subsequent stages. This uncertainty may suggest a misunderstanding of its application at this stage in the AV4AD model. As discussed in section 4.6.9, at this stage unit testing involves testers assessing potential modifications to the software component due to the identified defect which may require correction at earlier stages. Consequently, new tests must be developed for retesting the component. However, the implementation of the AV4AD model is expected to result in fewer defects at this stage when agile practices are proactively applied during earlier stages.

When asked whether the AV4AD model would assist MDS organisations in developing software while ensuring compliance with the IEC 62304 - Medical device software - Software life cycle processes standard, interviewee 1 responded that they would. They further suggested including the standard in the AV4AD model when they asked, "*So, it's complementary. So, I am not sure if you need or are you thinking of including the standard in this framework?*". However, the AV4AD model already integrates all the IEC 62304 lifecycle processes. In relation to their organisation using the AV4AD model and developing software while remaining compliant with IEC 62304 standard, interviewee 1 expressed confidence, stating, "*I believe so. Yes, it's similar to what we have now. So, I would think so"*. This statement was in reference to the V-Model currently followed in their organisation.

Regarding whether they would adopt the AV4AD model in their organisation, interviewee 1 affirmed that they would implement it, stating: "*Yes, this would align with the values we have going forward, and so it would be interesting*". Further, when asked whether implementing the AV4AD model in their organisation would assist in addressing defects, interviewee 1 asserted that it would. They stated, "*Yes, it would. Particularly, I think at the requirements stage using BDD and TDD earlier would certainly be very beneficial to identify those defects earlier*". When asked about the potential to conduct validation of the model in their organisation, they affirmed the possibility for the validation and recommended contacting their CEO for further discussion.

5.2.2.2 Interviewee 2 Feedback

Interviewee 2 discussed each agile practice at the stages of the AV4AD model, emphasising that they would address defects. Focusing on the left side stages and the coding stage of the model, they asserted that the agile practices in these stages would prevent defects. They further commended the complementary implementation of the practices in these stages, stating that it would enhance defect prevention.

In discussing agile practices in the left side stages of the AV4AD model, interviewee 2 affirmed their importance for addressing defects. Specifically, they discussed Model Storming, PP, and TDD for preventing defects, emphasising, "*model storming, pair programming, and test-driven development are massive*". However, they noted that in their current development approach, agile practices are not used in the early stages. Specifically, they mentioned user story, stating, "*we don't do user stories at the requirements phase to be honest with you*". Additionally, interviewee 2 discussed the use of onsite customer/proxy in the early stages, stating, "*it's probably too early to bring in customer on the left-hand side*". This statement is in relation to their current development process in which customer involvement is limited to later stages when there is something to test. They stated, "*So, we are over on the right-hand side, really. And you know it's coming up that side where we are going to be actually, you know, have something to go to a customer*". These findings suggest their organisation's focus on defect identification. Nevertheless, interviewee 2 emphasised the importance of onsite customer for addressing defects, stating, "*as I see it, the customer feedback at the requirements stage is vital really to addressing defects*". They also stated that they would use user story for preventing defects in their organisation, noting, "*that would be a change*" and "*it would be something that should be trialled. I do think that at requirements gathering user stories definitely help*".

At both the software architectural design and detailed design stages, interviewee 2 suggested including integration test planning to support TDD, particularly when TDD is used at the software unit implementation stage. They stated:

> "*I am suggesting at the architectural design and the detailed design, I would have integration test planning at those phases and then if you do use test driven development for your actual implementation, you have a very good head start on that*".

However, integration test planning is already considered at these stages as an initial step within the implementation of integration testing, where integration testers outline how different system components will be combined and tested together.

Examining agile practices in the right-side stages of the AV4AD model, interviewee 2 affirmed their suitability for addressing defects. They specifically mentioned unit testing, integration testing, code review, continuous integration, and refactoring, emphasising that these practices have been "*absolutely a huge driver of quality*" in their project. When asked about adopting the AV4AD model and developing software while ensuring compliance with the IEC 62304 requirements, they affirmed they would. They noted that the stages of the AV4AD model closely align with their current development approach, stating, "*we do everything that you are seeing here in the grey circles. So, it's quite similar*".

Regarding improvements to the AV4AD model, interviewee 2 recommended the inclusion of the IEC 62304, stating, "*I would definitely try and bring that in here because everything we do in medical has to conform to that for software*". However, the AV4AD model already encompasses all the IEC 62304 lifecycle processes. When asked about the possibility of validating the model in their organisation, interviewee 2 affirmed the potential for the validation and stated that they would consult their manager about it and provide feedback to the researcher. They said, "*Yeah. I mean, let me talk to my manager about that. I know they are very open to us partnering with external people, external you know, research. I will follow up with that, and I will come back to you*".

5.2.2.3 Interviewee 3 Feedback

Interviewee 3 discussed the AV4AD model focusing on agile practices at each stage, emphasising their suitability to address defects. Considering the left side stages and the coding stage of the AV4AD model, interviewee 3 stated that practices in these stages make the model

suitable for preventing defects. However, they noted that while certain practices like user story prevent defects, their implementation might be hindered by the lack of mandate in standards. They advocated mandating practices that address defects in MDS development. Despite acknowledging the importance of PP in preventing defects during requirements analysis, interviewee 3 was undecided about using this practice. They stated, "*Well, I think you know my opinion on pair programming. I am on the fence.*". Their perception on PP appears to stem from concerns about financial costs in engaging pairs of expert developers or testers. They elaborated, "*To get one person as the programmer on a project is wonderful. To request two there would be a lot of kickback from the industry*". However, the benefits of PP in addressing defects may ultimately justify its initial costs.

Regarding the right side of the model, interviewee 3 remarked that practices were suitable for identifying defects. Specifically, when discussing practices at software system testing stage, they stated "*I think you have reflected it well …*". Additionally, they emphasised the importance of Onsite Customer/Proxy throughout all stages but appeared to suggest that getting the actual customer involvement may be challenging in MDS development. They commented, "*Having the customer included throughout the process is obviously ideal, but in medical devices, and I do appreciate you have said proxy so it's someone standing for them which is good*." However, despite affirming its benefit in addressing defects, they suggested that this practice may not be valuable at the software unit testing stage. This perspective contrasts with the intended concept of proxy role which could include quality assurance personnel within the organisation validating the tested software unit. Through this involvement, defects could be identified.

When asked about potential improvements to the AV4AD model, interviewee 3 emphasised the importance of integration testing at the early stages, specifically the requirements stage. They stated, "*if you don't have integration testing perspectives at the requirements stage, it's for the most part too late*". However, this consideration had already been addressed through BDD at the requirements analysis stage.

When asked whether the AV4AD model would assist MDS organisations in developing software compliant with the IEC 62304 standard, interviewee 3 responded, "*yes, absolutely they would*". Overall, interviewee 3 commended the AV4AD model's potential in addressing defects in MDS development and expressed interest in adopting it in their organisation.

5.2.2.4 Interviewee 4 Feedback

Interviewee 4 evaluated the AV4AD model focusing on how their organisation could benefit from using it. They examined agile practices at each stage commenting on how they would assist in addressing defects whether used individually or complementarily. They affirmed that the practices are appropriate for addressing defects in the various stages they are included. They stated, "*This is a fine technique. There is no problem I see with this.*"

In discussing practices at the requirements gathering stage, interviewee 4 affirmed their suitability to prevent defects, stating, "*So, you are capturing using user stories here and then you are creating use cases to verify with customers, that's kind of an iterative process and that absolutely helps prevent defects*". Regarding agile practices at the software requirements analysis stage, they remarked, "*I don't see any problems with that. At this stage, they're all achieving the aim of basically telling you, giving you some more detail in your requirements*". This observation suggests the potential of agile practices at this stage to enhance clarity and prevent requirements defects.

Interviewee 4 acknowledged the importance of onsite customer/proxy in preventing defects by providing feedback during the software architectural design stage. However, they observed that in their current approach onsite customer/proxy "*may have not been possible until the full development cycle was completed or after software system testing*". This observation reflects the emphasis of their current plan-driven development approach on defect identification after development is completed. When discussing practices at the software detailed design and the software unit implementation stages, interviewee 4 stated that "*all the practices help towards defects minimisation*". In reference to practices at the software detailed design stage, they remarked, "*all those things are great*" while describing practices at the software unit implementation stage as "*all good*". Regarding the complementary use of practices for addressing defects at the software unit implementation stage, interviewee 4 stated, "*all of these are very important, like refactoring along with continuous integration, so that you are ensuring that anything that you do change doesn't have any unintended knock-on consequences*".

When discussing agile practices at the software unit testing stage, interviewee 4 stated as follows:

> "*Yeah, it's all good. Essentially, at this stage, you know your software unit is complete, you run your unit testing. That's either a pass or fail type scenario. If it's*

> *failed, you are going to do some change, some pair programming, or some other type of programming to fix. If the tests are sound and the unit is like fundamentals are fine, you are going to do some changes in running unit tests again. I think this is all fine*".

When asked about potential improvements to the AV4AD model, interviewee 4 noted that the model includes important practices for addressing defects and appropriately reflects their complementary use. They stated: "*Essentially you have captured at every stage here in the process the things that can be done and should be done, you know in this order as well. The ordering is important*". However, they emphasised the importance of including risk management at each stage. They explained:

> "*One of the main things with the medical piece of the software is this idea of risk management. So, there should be a piece coming out of each, say out of requirements gathering, out of analysis, out of architecture, that essentially is trying to identify risk at each stage*".

This perspective is in relation to the ISO 14971- Medical devices — Application of risk management to medical devices standard and suggests the need for developers to familiarise themselves with this standard and ensure their activities throughout the development lifecycle are validated against it.

Interviewee 4 further noted that adopting the AV4AD model would enable MDS organisations develop software while ensuring compliance with regulatory standards specifically the IEC 62304. They appreciated the model's emphasis on both defect prevention and identification throughout the development lifecycle. Additionally, interviewee 4 proposed integrating the ISO 14971 standard in the AV4AD model. They explained that including this standard would enable the model serve as a guiding framework, ensuring MDS developers become familiar with risk management within "*a larger quality management system … during development*". Interviewee 4 further emphasised the importance of including the IEC 62366-1 - Medical devices - Part 1: Application of usability engineering to medical devices standard in the AV4AD model. They explained that by including this standard the model would serve as a guiding framework, enabling MDS developer's familiarisation with its requirements and develop usability evaluation plans and protocols for the acceptance testing stage. When asked about adopting the AV4AD model in their organisation, interviewee 4 affirmed, "*Yes, we would*". They further emphasised the inclusion of risk management into the model. When asked

about the potential to conduct the validation of the AV4AD model in their organisation, they affirmed their openness to conducting the validation process.

5.2.2.5 Interviewee 5 Feedback

Interviewee 5 discussed the AV4AD model focusing on the suitability of agile practices to address defects in the stages they are included in, and the model's fitness for MDS development. When discussing agile practices at the requirements gathering stage, they affirmed all practices for defect prevention relating them to how they have been used in their current development approach. Specifically, they emphasised the importance of onsite customer/proxy in defect prevention. They stated: "*So, certainly they would be key channels and particularly some of our power users or power customers would be very proactive in identifying current bugs or flagging potential bugs that maybe we need to investigate*".

When discussing practices at the software requirements analysis stage, interviewee 5 commented that they are appropriate for preventing defects. Specially, they emphasised the importance of BDD for this purpose and mentioned its adoption in their current development approach. However, they acknowledged that model storming has not yet been adopted. They stated:

> "*I don't know much about model storming, to be honest, but we have gone down the BDD route. We have not fully implemented it, but out of what you've listed there, I would say BDD is our favourite approach and has the most promise as well*".

Interviewee 5 also emphasised the importance of TDD during the requirements analysis stage for creating tests early before writing the business code, stating,

> "*Yes. So, they would really be writing the tests before writing the business code. And those tests are a key component or key part of our build and release pipelines in Azure DevOps*. *These tests are linked to the user story which is linked to the code that's deployed to production".*

When discussing agile practices at the software architectural design stage, interviewee 5 noted that they are suitable for preventing defects. However, they reiterated that model storming is not adopted in their organisation. Regarding practices at the detailed design stage, they affirmed their importance for addressing defects. They specifically mentioned that Onsite

Customer/Proxy "*more generally would be the most powerful or most effective way of identifying bugs, errors and defects during the detailed design stage*". Additionally, they emphasised the complementary use of onsite customer/proxy and model storming to further enhance defect prevention. Furthermore, interviewee 5 recommended including BDD at this stage, stating that "*BDD is more detailed and more closely aligned with the software detailed design*".

In discussing practices at the software unit implementation stage, interviewee 5 emphasised their importance in addressing defects. Regarding TDD, they remarked, "*So, following TDD would help us identify what those potential bugs, errors, the defects are, and to cover them*". They also underscored the importance of adhering to coding standards to prevent defects. Regarding unit testing, they stated that "*Unit testing absolutely is key. Having those unit tests and other tests that run every time we run a build and release, whether it's preproduction or production is vital*". Additionally, they affirmed refactoring as important in dealing with technical debt and was beneficial when they moved to a new three-tier version of their software. Interviewee 5 further noted that their organisation uses small release and helps in addressing defects. Regarding CI, they commented, "*that's a fundamental part of how we work"* and that it addresses defects. However, interviewee 5 explained that code reviews in their organisation are only conducted when there is a quality issue. They stated: "*We generally only do code reviews if there are quality issues going on and we need a senior person to step in for a while and start to do some code reviews, but we don't do all the time*".

In relation to practices at the software unit testing stage, interviewee 5 affirmed their suitability for addressing defects. They described unit testing as "*the most basic unit of protection against any new bugs and defects*". Additionally, they reiterated the benefit of refactoring in their current development approach. Regarding the practices at the software integration and integration testing stage, interviewee 5 noted their importance for addressing defects. However, they suggested that UAT at this stage and subsequent stages should encompass usability testing and accessibility testing. They explained, "*Well, it all depends on what your definition of user acceptance testing is, but I would say that you have usability or accessibility testing within your definition of UAT*". This perspective is in relation to the ISO 62366-1 and WCAG-2 standards which they said have become increasingly common in the MDS domain. They noted that clients frequently consider whether the product satisfies these standards when deciding to

purchase MDS. Consequently, they suggested the inclusion of these standards in the AV4AD model for guidance to developers.

During the discussion on practices at the software system testing stage, interviewee 5 affirmed their suitability for addressing defects. Additionally, they suggested the inclusion of AI models, mentioning that their current development approach integrates a "*prediction model which uses logistic progression*" for anomaly detection. In relation to practices at the acceptance testing stage, interviewee 5 noted their suitability for addressing defects. However, they were unsure about PP as it is not used in their organisation. They emphasised the importance of UAT and recommended both manual and automated UAT at this stage, stating, "*So, UAT would be manual and automated and the automated UAT is important*".

Interviewee 5 noted that the AV4AD model is suitable for MDS development while ensuring compliance with the IEC 62304 standard. When asked about adopting the AV4AD model in their organisation, interviewee 5 stated:

> "*Sure. Yeah, absolutely. Yeah. It's good timing and I certainly would like to use it. Even looking at this, there's a few gaps that I know are there in our organisation and this would be a good rubric to use*".

When asked about the possibility of validating the AV4AD model in their organisation, interviewee 5 agreed and indicated openness to further discussions about scheduling the validation.

5.2.2.6 Interviewee 6 Feedback

Interviewee 6 examined the AV4AD model and discussed agile practices at each stage in relation to their adoption within their organisation. Regarding practices at the software requirements gathering stages, they commented that they are suitable for preventing defects. They explained that their organisation uses a combination of user stories and onsite customer stating, "W*e are using user stories. And it's a combination of both user stories and onsite meetings with customers where we have like meetings with them and workshops to try to identify as much as possible*". However, they indicated that use cases are not currently used in their development process.

When discussing practices at the software requirements analysis stage, interviewee 6 affirmed their potential to prevent defects. However, they noted that model storming and BDD are not

used at this stage in their current development approach. They expressed interest in implementing these, stating, "*So, at the software requirements analysis, we don't go to the level of storming like model storming or behaviour-driven development. That's probably something that I would like to get to do*". They further explained that PP is only used to support cross-team collaboration especially when a team member must work in an area controlled by another team or when explaining the code to new or junior developers. Additionally, they clarified that this approach ensures that team members become familiar with other teams' activities before they can perform any task, ultimately contributing to defects prevention. Regarding TDD, interviewee 6 mentioned that it is adopted in their organisation and has been useful in preventing defects. They stated, "*We do use TDD a lot and it's something that we enforce. And from my experience, I think that's one of the best*". *TDD is very helpful in preventing defects. Because it helps developers not make assumptions*".

Interviewee 6 affirmed practices at the software architecture design stage for preventing defects. They emphasised the benefit of integration testing at this stage in their development approach, specifically the creation of integration tests which they said helps to prevent defects. They stated, "*We do use integration testing, designing the integration test is a big part of that. And I do feel that it's very good to prevent any defects*". Regarding onsite customer/proxy, they commented, "*we do involve customers at all stages*". They further mentioned that in their organisation the proxy role is handled by the quality assurance engineer. Additionally, interviewee 6 mentioned that model storming is frequently used at this stage in their organisation.

During the discussion on practices at the software detailed design stage, interviewee 6 said that they are suitable for preventing defects. They emphasised the importance of integration testing at this stage, stating, "*it is very important to have it at this stage, and it is very important for preventing bugs*". They further supported the complementary use of practices to enhance defect prevention. They emphasised the importance of involving customers throughout all stages of development to prevent defects that arise due to misunderstandings among stakeholders. They explained:

> "*So, combining them is very important. It's very important even during requirements analysis and gathering. We do find a lot of time, a lot of the cases that there are misunderstandings between what the customer wants and what the product team and what the developers understand. So, having the customers*

> *involved at every step is important because you don't go too much into implementing something wrong before the user could approve it*"

Regarding practices at the software unit implementation, interviewee 6 stated that the practices address defects and have been adopted in their organisation. They reiterated the importance of integration testing and mentioned that TDD is enforced in their development approach. Regarding code review, they underscored the importance of involving team members who are more familiar with the code base. They explained,

> *"We do enforce code reviews and approvals, so at least 2 developers need to review the code and usually we find there are loads of cases where defects are even visible from the code review because we usually try to ask the people who know the code base better to do the code review"*

Interviewee 6 further emphasised the importance of code ownership and coding standards in addressing defects in their organisation. They stated, "*code ownership is one that I found important and coding standards as well*". Elaborating on code ownership in their organisation, they explained, "*we do have ownership of areas of the product, and usually if some features that we need to implement would touch other teams areas, we are normally asking other tech leads or other developers to do code reviews*". Regarding small release, interviewee 6 mentioned their internal benefits, stating, "*So, what I found useful for this small release is they are useful to have them internally. So, we do have like a nightly we call it a nightly release*".

At the software unit testing stage, interviewee 6 expressed that the practices are suitable for addressing defects. They specifically discussed the nature of the unit testing process within their development approach. Regarding practices at the software integration and integration testing stage, interviewee 6 affirmed their importance for addressing defects and discussed the integration testing process in their organisation. They also affirmed the use of PP at this stage and explained how pairing between automation engineers and developers contributed to improved product quality. They stated:

> "*Yes. When automation engineers write the tests, we usually do have sessions with them to explain some of the things or help them with the features. So, again, it's something to speed up the process of writing automation tests and making them reliant. We only recently started to do that, and we did find out that the quality of our product did improve*".

Additionally, interviewee 6 explained the role of refactoring at this stage, particularly in enhancing performance of the product. They explained, "*Refactoring at this stage is mostly to do with performance. So, usually if performance is affected, we do go back to the step and we need to send back the code to be refactored*". They further highlighted the importance of integration testing and CI for identifying defects at this stage. Regarding CI, they stated that "*continuous integration is very important*".

Discussing practices at the software system testing stage, interviewee 6 mentioned that the practices are suitable for addressing defects. However, they noted that in their organisation, PP and refactoring are not used at this stage and the acceptance testing stage. Instead, defects that necessitate changes to code are referred to the appropriate earlier stage for resolution. They stated:

> "*We would usually just go back to software detailed design. So, usually if a defect is found, we would create a new development task to fix that defect, go through every stage, from the software implementation, write unit tests for it, make sure the coding standards and everything that is involved at that step*".

When asked about additional practices they would like to be included at the software system testing stage, interviewee 6 suggested implementing code ownership. They explained that this would help to ensure that the identified defects are traced back to the relevant development team or owners of the code/module or unit for resolution. They stated:

> "*I think ownership of those defects would be important to know. Like if we do have code ownership or feature ownership for the product level, would be usually important to send it back to the exact team that would have the knowledge*".

At the acceptance testing stage, interviewee 6 emphasised the importance of UAT and onsite customer/proxy for addressing defects. They mentioned that often defects that slip through the earlier stages can be caught at this stage. They explained, "*A lot of the times some defects that nobody thought about could be found in that stage. So, yes, it is important to have that level of testing before it can go out to everyone*".

Overall, interviewee 6 stated that the AV4AD model is suitable for MDS development and addressing defects while conforming to international standards such as the IEC 62304. They commended the AV4AD model's flexibility to allow developers to address defects at any stage in the development lifecycle. They stated, "*That makes sense. So, even on the software unit*

testing that you have there, if you find a defect, so that needs to go back to the detailed design and implementation, make sure that you correct it". Interviewee 6 expressed interest in adopting the AV4AD model in their organisation, emphasising its ability to guide developers in selecting and using appropriate agile practices for addressing defects throughout the development lifecycle. They stated, "*Yeah, definitely. This would be a good methodology to follow"*. However, they noted that their organisation currently follows the SAFe framework. Considering the huge financial investment made in its implementation, they believe management would be reluctant to switch to a different approach.

5.2.2.7 Interviewee 7 Feedback

During the interview, interviewee 7, a senior software engineer and AI specialist, examined the AV4AD model focusing on the agile practices included at each stage and their use in addressing defects, as well as its overall structure for guiding the software development process. Their foremost response was the commendation, noting that following the AV4AD model could have been beneficial in their project. They stated, "*You know, all of this, I look at it and I was like, God, I wish we did that when we first started developing*". Regarding the agile practices at the requirements gathering stage, interviewee 7 commended the inclusion of onsite customer or proxy and user story for preventing defects in the development process. They stated:

> "*Definitely going for the user story and having the onsite customer or proxy I think is very good practice. So yeah, I am quite a fan of like really having the user be integral to your entire development process instead of just developing it and then trying to get a user to fit into what you have developed. You know, I like that user-centric focus*".

Reflecting on their project, interviewee 7 mentioned that user story had not been used and emphasised the importance of both user story and use case for addressing defects at this stage. They stated:

> "*Having the user story there is, yeah, I think is vitally important. I don't think we had the user story totally in mind when we first started developing. But yeah, I think that's very important and use case as well. Again, if these things are as clearly defined as they can be from the onset of development, I just think that's vitally important*".

When asked about any additional agile practices that could be used for addressing defects at the requirements gathering stage, interviewee 7 expressed confidence in the practices already included and reiterated their importance for addressing defects at this stage. They stated, "*No, for this stage, no. I think having that as your start point, I think, is the most important and focusing on that, I think, is the right thing*".

At the requirements analysis stage, interviewee 7 affirmed the relevance of the agile practices for preventing defects and commended their complementary application in enhancing defect prevention. They stated, "*You start the model storming, behaviour-driven development and then you are like verifying that a bit with paired programming and a TDD and then you still have your on-site customer in mind. Yeah, I like that approach*". In response to a question about potential improvements in addressing defects at this stage, interviewee 7 suggested conversations with large language models, stating "*thinking about the requirements you need with a large language model*". However, this suggestion falls outside the scope of this research. When asked about any additional agile practices for addressing defects at this stage, interviewee 7 affirmed the ones included and reiterated their complementary application. They stated:

> "*No. But I do really like this approach in general. I like, you know, the idea of model storming, behaviour-driven development, and I like that there is like verification kind of processes there. Yeah, and again, just that you keep mentioning the on-site customer, I think is very important*".

Regarding the agile practices at the software architectural design stage, interviewee 7 affirmed their importance for preventing defects and commended their complementary application. They stated: "*I think that's very good practice in catching defects*". However, interviewee 7 was initially uncertain about the simple design practice at this stage and requested further clarification. After clarifying, they confirmed its applicability and suggested that simple design should be implemented prior to integration testing. They stated:

> "*I really like the thinking about this like it's all lowering the probability of defects. I really like the idea of that, yeah, with the simple design maybe come before, like the first integration testing. Because I like the idea that you model storm, you have the onsite customer in mind, you come up with this initial simple design and then you like, you know, integrate, do integration testing just to see in this simple design, do all these parts fit*".

The suggestion to have simple design before integration may be a misunderstanding of the complementary application practices at this stage, particularly integration testing, which, as mentioned previously, involves integration testers throughout the software architectural design process.

When asked about additional agile practices that could address defects and any potential improvements at the software architectural design stage, interview 7 commended the practices already included and their application, stating, "*I do really like the process behind all of this*". When further asked whether these agile practices could address defects at this stage, they affirmed, "*Yeah, definitely, definitely they would*".

Interviewee 7 affirmed the importance of the agile practices included at the software detailed design stage for addressing defects and their complementary application. They particularly emphasised the application of simple design, model storming, and onsite customer. In relation to simple design, they indicated that its use could have prevented interface defects encountered in their development process. Like in the previous stage, interviewee 7 reiterated that simple design should be implemented first at this stage. They stated:

> "*So yeah, again, I kind of like the idea that, let's say you're developing the front end, that you also have like, come up immediately with a simple design there, because I know we didn't do that from our own front end and it got very messy, so I like that. You know this idea of like come up with something simple first, you know, along with the model storming process and keeping your on-site customer there. I really like this. Yeah, maybe the same comment of like the simple design for me could be more forefront. Other than that, I really like it*".

When further asked whether the agile practices included at this stage could address defects, they affirmed, "*Yeah, yeah, they would*".

At the software unit implementation stage, interviewee 7 commended the agile practices included for addressing defects. They began by providing their overall opinion, stating, "*Yeah, no, I mean it all seems, just from the outset here, overall, it seems very good to me, and I couldn't think of more agile practises to prescribe to that*". Interviewee 7 then emphasised the importance of code review, coding standards, code ownership, unit testing, refactoring, and small release for addressing defects at this stage. They stated:

> "*Yeah, Code review … I do think, yeah, that's quite important. … So even if you're developing with AI, I think like code review is quite important. …. So, I think maybe even code review is more important now than ever. I think it's very good you've coding standards and code ownership in there that's really important. You know, you don't just implement something without these things in mind. I think that's very good. Yeah, unit testing, small release, very important, refactoring, code review, like this cycle, I think. … I really like that all these things are there and implemented and yeah, you do that for each unit you implement*".

Regarding agile practices at the software unit testing stage, interviewee 7 affirmed their importance for addressing defects at this stage. They particularly emphasised refactoring and suggested the inclusion of code review to enhance defect identification. They stated:

> "*Hopefully, true refactoring, you get a more kind of, let's say, condensed code; it's easier to understand, it's, you know, works better, and so all those things are great. Code review should be there as well or something like I feel like that's a consequence of refactoring that it's almost a code review in a sense*".

Their suggestion to include code review at this stage may reflect a concern that refactoring, if performed incorrectly could introduce new defects. However, as discussed in chapter 4, this risk is already mitigated at this stage through unit testing, where any refactoring is followed by unit testing.

At the software unit integration and integration testing stage, interviewee 7 affirmed the agile practices included for addressing defects at this stage. They stated: "*Yeah, so Integration testing, PP, refactoring, unit testing. I think all that is really good. Continuous integration as well. Yeah, and you have the on-site customer and your team. Yeah, … I think that's quite good*". Interviewee 7 further commended the complementary application of these practices at this stage. They stated:

> "*I think those specifically would be very good at identifying defects. You know, it's like multi-kind of factorial approach. The more I think of refactoring, you reduce the risk of defects, to unit testing, you reduce the risk of defects, and I think altogether you have a combined effect of reducing defects, so I think it's quite good*".

With respect to agile practices at the software system testing stage, interviewee 7 affirmed their inclusion for addressing defects. They particularly emphasised UAT for addressing defects at this stage, stating "*I really like that. ... I think the UAT is great to have there*". Interviewee 7 further asserted the complementary application of the practices to address defects at this stage. They stated that "*...combining UAT with the paired development and refactoring and the unit testing is a strong approach*". At the acceptance testing stage, interviewee 7 commended the agile practices included for addressing defects at this stage. They stated:

> "*I think for acceptance testing, it just has to be UAT led, and then I like that you have the like secondary check almost of PP there, and yeah, you still have on-site customer or proxy there. So, I think it's quite good as is*".

When asked whether MDS organisations adopting the AV4AD model would remain compliant with standards including IEC62304, interviewee 7 confirmed that they would. They stated:

> "*Yeah, it seems to me that they would. Yeah, it seems to be that at each stage, aligns with something there and, yeah, it's very useful to do this sort of practise so you're immediately in line and you don't have to be kind of retrospectively in line. You don't have to come back after the fact and try to be in line. I think this, yeah, this does align with it*".

Regarding adopting the AV4AD model in their organisation, interviewee 7 expressed strong support for adoption the model and indicated that they would recommend its use to their coworkers. They stated:

> "*Yeah, I would. I would definitely adopt it. I think. We really lacked a lot of these processes, and I wish we had, you know, done them at the time, so going forward, I would. I would adopt this workflow for sure. I would recommend it to my coworkers as well*".

Overall, interviewee 7 commended the AV4AD model for addressing defects and said that MDS organisations adopting it would remain compliant with established standards. They stated:

> "*Yeah, I think it's really strong for addressing defects and I do really like that there's, I think at every stage, multiple ways of catching defects, which I think is one of the strongest elements of it and it's clear. And, yeah, I think it's a strong model for not only catching defects but preventing defects in the first place. I think*

that's, yeah, a core strength of it. I think it clearly demonstrates how you can prevent defects in in the development process of medical software and has the joint effect of being in line with established standards."

5.2.2.8 Interview 8 Feedback

Interview 8 involved two participants, a software engineer and a quality specialist. They both examined the AV4AD model focusing on addressing defects and the overall software development process. Their immediate observation was that the planning stage is not included in the AV4AD model. As noted earlier, the AV4AD model stages are adopted from the IEC 62304 standard, in which planning is considered a prerequisite activity rather than a software lifecycle process. This was further corroborated by the quality specialist when they stated that "*identification of technologies and feasibility and all of that kind of stuff that's done outside of IEC62304*".

At the requirements gathering stage, both interviewees affirmed the agile practices included and their complementary application in addressing defects. The software engineer particularly commended the iterative nature of applying the practices to address defects at this stage. They stated:

"*… requirements on user story, onsite customer, yeah, and yeah, just really iterative process. .... that's gonna have to be iterating a lot before you move on ... and yes, I agree with that, yeah. And the user story would definitely be a different step to the use case because the story is only understanding the story and then finding what the use case actually is another part of the journey*".

When asked whether any other agile practice should be included for preventing defects at this stage, both interviewees expressed confidence in the ones already included. The quality specialist said that they "*could not think of any others*", and the software engineer similarly noted that "*it's all good*".

At the software requirements analysis, the software engineer affirmed the agile practices and expressed confidence in their effectiveness for preventing defects at this stage. They stated that the practices were "*all good*" and that there was "*nothing to add ...*". The software engineer further endorsed the complementary application and the direction of implementation of the practices for preventing defects at this stage, stating that the approach "*is good*".

Regarding agile practices included at the software architectural design stage, the software engineer affirmed their use for addressing defects. In particular, regarding onsite customer, they commented that it "*needs to be a requirement, definitely*" at this stage. They further noted that "*definitely simple design at the start*" is important. However, they expressed uncertainty about integration testing at this stage. After clarification about its application, the software engineer endorsed its use at this stage. They stated: "*Yeah, yeah, yeah, that would be great. Yes, yeah. I agree with that*". When asked about other agile practices for addressing defects at this stage, both participants expressed confidence in the ones already included. The engineer stated that the practices were "*all good*".

At the software detailed design stage, both participants affirmed the agile practices for addressing defects. In particular, the quality specialist recommended that "*pair programming, and refactoring with someone else independently is very helpful*". The software engineer specifically commended small release and refactoring, stating, "*Yeah, small release, absolutely. That's small changes, testing, small changes, testing, small changes, testing. Refactoring is obviously something that might be needed, but you'd hope it's planned well enough*".

However, both participants suggested combining the software detailed design, software unit implementation, and software unit testing stages into a single stage. They stated that they view these stages as one "big circle" rather than distinct stages. While this perspective is in line with the iterative mindset, merging these stages without clearly delineating their boundaries may undermine essential checks and balances of the SDLC. It risks the loss of critical design detail, increases the likelihood of omission defects, and may introduce other longer-term development risks. In particular, overlooking or diminishing the visibility of the detailed design stage may encourage developers to jump straight into coding. This may increase the risk of coding defects, technical debt, and architectural inconsistencies.

Both participants affirmed the agile practices included for addressing defects at the software integration and integration testing, software system testing, and acceptance testing stages. However, the software engineer suggested that refactoring should be limited to the detailed design, unit implementation, and unit testing stages. However, this perspective may be due to a misunderstanding of including refactoring to correct miner defects at these stages as discussed in chapter 4.

5.2.2.9 Discussion

The interviews yielded very positive results regarding the usefulness of the AV4AD model in addressing defects within MDS development. Interviewees discussed the importance of practices at each stage of the model and their complementary applications, emphasising its potential to address defects in their organisations. The evaluation results further indicate that the AV4AD model is suitable for MDS development and could help MDS organisations in addressing defects.

Additionally, the results reveal the diversity in how organisations adopt agile practices in MDS development. For example, while interviewee 5's organisation uses model storming at the requirements analysis stage, interviewee 6's organisation does not. This diversity underscores the necessity for agile models in MDS development to offer flexibility, allowing developers to select relevant agile practices to address defects at any stage in the development lifecycle. The AV4AD model inherently offers this flexibility, ensuring defects can be addressed throughout the development lifecycle.

All interviewees affirmed the AV4AD model's flexibility, particularly its ability to allow defect correction at earlier stages before progressing. They also confirmed the model's alignment with regulatory standards, particularly the IEC62304, which defines lifecycle processes for MDS. Furthermore, interviewees affirmed the model's flexibility in allowing developers to select individual practices or a combination of practices allowing for complementary use at a particular stage to enhance defect prevention or identification.

The findings from the semi-structured interviews indicate that the AV4AD model is suitable for addressing defects in both SMEs and large organisations. The model's emphasis on both defect prevention and identification throughout the development lifecycle, coupled with the flexibility to correct defects at any previous stage before progressing, makes it valuable for MDS development. The model not only offers the potential to minimise costly rework but also reduces the likelihood of defects in the final release. Moreover, all interviewees affirmed that the AV4AD model would enable organisations to develop MDS while ensuring compliance with international standards, particularly IEC62304. The results showed strong endorsement for future implementation of the model in their organisations. Although some organisations indicated limited prior use of practices such as BDD and model storming, all interviewees

acknowledged the importance of these practices in addressing defects. Moreover, they stated their intent to implement these practices in their future projects.

Regarding potential improvements to the AV4AD model, interviewees primarily suggested the inclusion of standards, specifically ISO 14871 - Medical devices — Application of risk management to medical devices, ISO 62366-1 - Medical devices - Part 1: Application of usability engineering to medical devices, and WCAG-2 -Web Content Accessibility Guideline 2, to serve as a reference for developers. Additionally, interviewees 1 and 2 suggested including IEC 62304 - Medical device software – Software life cycle processes. However, this standard is already considered within the AV4AD model as its stages align with the lifecycle processes outlined in IEC 62304.

Apart from standards, participants in interview 8 suggested combining detailed design, unit implementation, and unit testing into a single stage. However, as discussed earlier, this perspective may undermine checks and balances in the model, increasing the risk of coding defects and omission defects which may lead to expensive rework. Interviewee 5 suggested the inclusion of security-driven development and integration of AI models within the AV4AD model. Following a careful assessment, these suggestions were deemed to fall outside the scope of the present study. Both security-driven development and integration of AI models in MDS development represent broad areas requiring dedicated research. Future research could use the AV4AD model as foundation to explore these areas. Therefore, only the suggestion to include standards was addressed in this research. It is important to stress that these standards have been included in the model strictly for reference purposes, as compliance solely depends on the organisation's commitment to follow them. The next section details this decision and presents the final AV4AD model based on the evaluation feedback.

5.2.2.10 Inclusion of Standards in the AV4AD Model

As discussed in section 2.9, standards are integral to ensuring the reliability of MDS. Given the existence of a multitude of standards that relate to the safety and reliability of MDS, it can be challenging for new entrants to easily identify and familiarise themselves with those particularly relevant during the development lifecycle. Considering both this aspect and the evaluation feedback, specific standards that are crucial for maintaining reliability throughout the MDS development lifecycle have been included in the AV4AD model. The aim is to guide MDS organisations on which standards they can refer to and validate their processes against at various stages when using the AV4AD model. For example, during the requirements gathering

stage, it is important to consider the potential risks associated with each requirement and devise measures to deal with these risks. The IEC 14971 - Medical devices — Application of risk management to medical devices standard is particularly relevant at this stage, as it focuses on risk management. Early identification of risks and preparing measures to address them ensures that at each stage of development, the team checks if they have addressed the identified risk.

Similarly, the ISO 62366-1 - Application of usability engineering to medical devices and the Web Content Accessibility Guidelines -2 (WCAG-2) standards are indicated against the software system testing and acceptance testing stages to underscore the importance of validating activities against these standards. The WCAG-2 is an international standard developed by the World Wide Web Consortium (W3C). This standard aims to make web content more accessible to people with disabilities including visual, cognitive, physical, and hearing impairments (W3C 2008). Usability and accessibility are important for the acceptance of the developed product. Therefore, by including ISO 62366-1 and WCAG-2 within the AV4AD model, organisations are encouraged to refer to these standards when using the model to ensure that usability testing and accessibility testing are incorporated into their development processes.

As stated earlier, the inclusion of standards in the AV4AD model serves as a guide for developers to refer to these standards when using the model. However, their inclusion does not imply compliance, as compliance depends on an organisation’s familiarity with these standards and its commitment to following them. In other words, the standards are included in the AV4AD model to facilitate developers’ familiarisation with them, so that they can validate their activities against regulatory requirements during the development process. The final AV4AD model is shown in Figure 5.1.

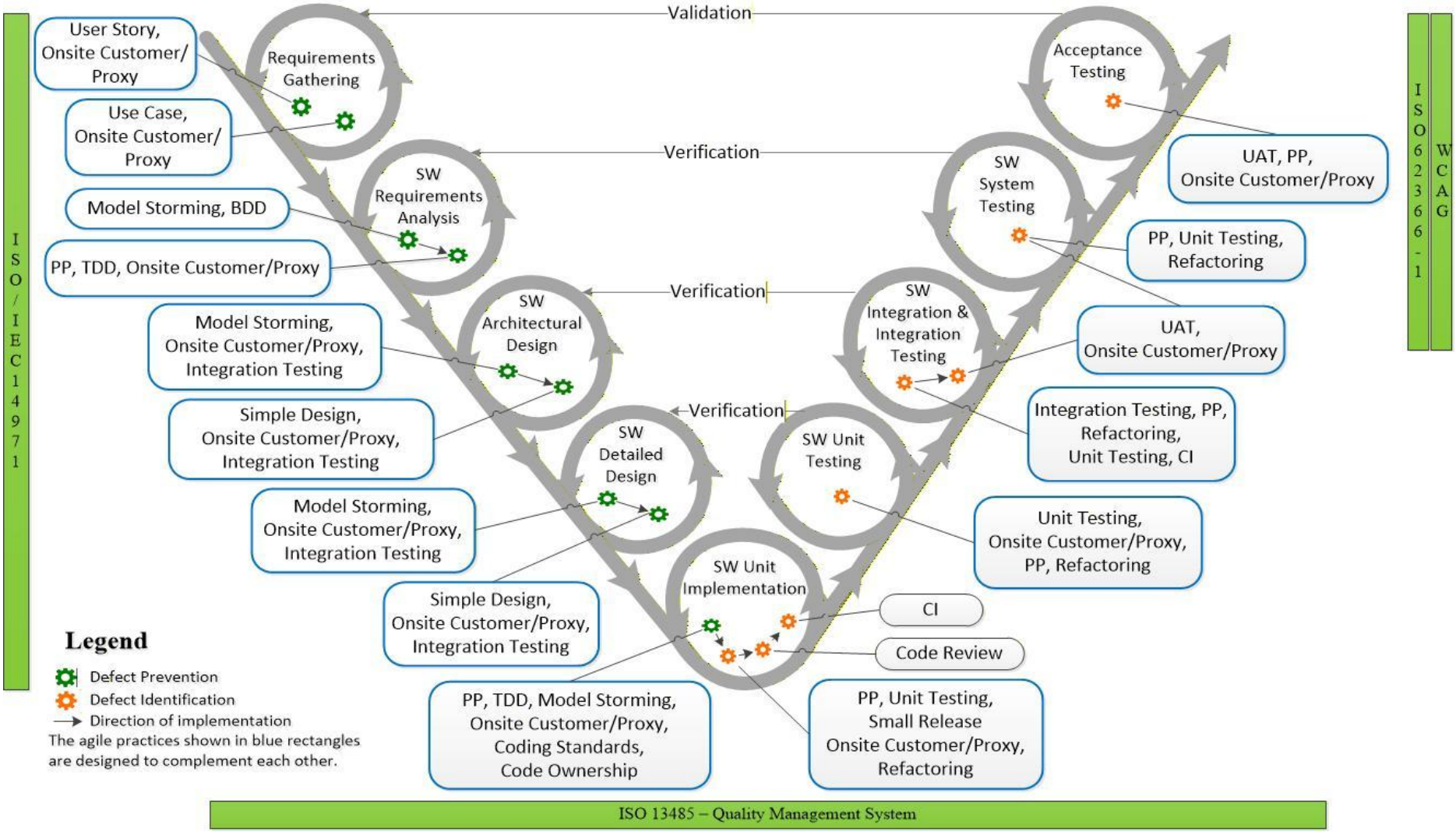


Figure 5.1: Final AV4AD model

5.3 Validation

5.3.1 Introduction

To further ascertain the efficacy of the AV4AD model's ability to address defects in MDS development, it was important to validate it in an MDS development project. After incorporating the improvements received from the semi-structured interviews, the AV4AD model was ready for validation. This section details the validation process and results.

5.3.2 Identifying a Participating Organisation

The key aspect in the validation process was identifying an MDS organisation that had a project and was willing to participate voluntarily. To maximise the chance of finding a suitable organisation, the request was opened to several MDS organisations in Ireland. During the evaluation process detailed in section 5.2, six participants stated that they would implement the AV4AD model in their organisations. The six organisations were then formally contacted to confirm their participation, project availability, as well as indication of a convenient time to conduct the validation. They were also requested to share any relevant information about the project for the researcher's familiarisation and planning purposes. To further increase the chances of finding participating organisations, additional outreach efforts were made. The Cluster Manager for the Connected Health and Well-Being Cluster, located within DkIT, was contacted to request MDS organisations within the cluster for participation. Further efforts were undertaken to broaden participation in the validation process by inviting 48 MDS organisations that did not initially participate in the evaluation process. Three organisations responded but only two agreed to participate. The third proposed to participate in the later months of the year 2026, but this fell outside the timeframe for this project.

Three SMEs that participated in the semi-structured interviews responded and initially confirmed their participation in testing the AV4AD model in a new project. However, scheduling proved challenging due to their tight deadlines which meant no staff could be available for the validation. Considering the limited time for this research project and the elapsed time without the validation schedule, an alternative validation approach was proposed. Instead of implementing the AV4AD model in a new project, the alternative approach involved a desk-based implementation on a past project. The aim was to retrospectively assess whether its application could have addressed defects.

The retrospective desk-based implementation approach offers a faster, more cost-effective, and efficient means of gathering insights compared to conducting a new project implementation (Lehtinen et al. 2015; Dingsøyr 2005). This approach typically involves analysing existing documentation like project plans and defect reports, as well as engaging in discussions with personnel that were involved in the initial project implementation. In this research, only the latter technique was employed as project documentation was unavailable due to both its absence and confidentiality constraints. The implementation process involves detailing how the initial project was executed, including defects encountered and the SDLC model followed. The subsequent analysis then examines how the proposed SDLC model would have been applied and evaluates whether it could have addressed defects and challenges observed in the initial implementation.

In total, four MDS organisations agreed to participate in the validation process. All four organisations welcomed the alternative approach, and arrangements were made for the validation. While the initial plan was to validate the model in several MDS organisations, participation was secured from only four, primarily due to the project's constrained timeline. The subsequent subsections provide descriptions of participating organisations and the validation process. To ensure anonymity and comply with ethical guidelines, the actual names of participating organisations are not used. Instead, the pseudonyms 'Organisation A' and 'Organisation B' are used in this research.

5.3.3 Validation in Organisation A

5.3.3.1 Profile of Organisation A

Organisation A, a spin off company from an Irish University, was established in 2007 and is headquartered in Ireland. The organisation specialises in point-of-care MDS, which falls in Class I category of the Regulation (EU) 2017/745 (EU 2023). Class I medical devices do not pose serious injury risk to patients. Previously Organisation A developed two types of medical products: hardware medical devices and software. However, the organisation phased out the hardware product and currently focuses only on the software product. At present, Organisation A has a total of six personnel that include the Chief Executive Officer (CEO), three developers, and two customer engagement specialists. Their software is designed for deployment in a client-server setup within the hospitals, with devices connected wirelessly. The software enables users such as nurses and clinicians to electronically manage patient records, perform reviews and conduct clinical assessments. Organisation A follows the Waterfall approach for

software development. Over the years, their software has undergone improvements through versioning. The version of the software currently in use was deployed in 2013.

5.3.3.2 Validation Process in Organisation A

The AV4AD model was implemented on the software version that preceded the currently deployed version. As mentioned in section 5.3.2, the implementation process was conducted retrospectively involving the researcher and one developer from the organisation who was familiar with the development of the software's version. The developer had participated in semi-structured interviews and was therefore familiar with the AV4AD model. They commended the model, stating "*this model has everything you need, covers all the important pieces"*.

The implementation process started with the developer providing the project background, detailing its origins and subsequent implementation using the Waterfall approach. They explained that the project originated from research conducted by a researcher at an Irish University to address a problem related to point-of-care within the healthcare in Ireland. Therefore, the potential client base (hospitals) had already been identified before the software was developed. However, the developer explained that initially no engagement was made with the clients until this version of the software was showcased to potential funders. Once the funders had accepted the proposal, collaboration was established between Organisation A and the first client hospital. Subsequently, the software was showcased to the client. The developer stated that the client hospital identified defects in the software which necessitated significant rework. The identified defects were specifically concerning requirements, interface, functionality, and performance.

The developer then discussed the project's implementation using the Waterfall, focusing on the outputs at each stage in relation to compliance with industry standards particularly IEC 62304. Thereafter, the developer and the researcher examined the project's implementation using the AV4AD model, comparing it to the original waterfall implementation in terms of addressing defects. Throughout the process, at each stage, the developer assessed whether all practices would have been used or if only a selection would have been used. This decision-making process aligned with the model's flexibility, allowing developers to use all or only selected practices at different stages. Additionally, the developer was asked to determine whether each of the implemented practices at a stage addressed defects. They were also asked to ascertain

whether among the implemented practices there was a practice(s) that worked better than the others in addressing defects at a particular stage.

5.3.3.3 Results from Organisation A

The developer emphasised the importance of early user involvement to provide feedback. Consequently, at the requirements gathering stage all practices were selected for implementation. They stated that this approach would have addressed requirements defects which in the previous implementation were identified when the software was initially showcased to the client. Similarly, at the software requirements analysis stage, all practices were selected for implementation. Reflecting on their previous implementation, the developer affirmed that using all the practices at this stage would have increased the likelihood of catching requirements defects early, minimising their occurrence in later stages.

At the software architectural design and software detailed design stages, the developer preferred a complementary application of onsite customer/proxy, model storming, and integration testing to prevent design and integration defects. Compared to their previous waterfall approach, they emphasised that active participation and collaboration among team members in roles such as developer, software architect, and integration tester, alongside onsite customer/proxy, would have ensured that the software's structure and behaviour aligned with client needs while also addressing potential design and integration defects early. Additionally, they stated that at these stages, the integration tester would have created verification tests for individual components and the overall software. These tests would have identified defects that escaped these stages.

For the software unit implementation, the developer selected PP, onsite customer/proxy, coding standards, code ownership, small release, code review, and refactoring. Although they acknowledged all practices were important for addressing defects at this stage, they preferred the selected practices for the project. Reflecting on their waterfall implementation, the developer affirmed that applying these practices would have addressed potential functional and performance defects in their software.

During the software unit testing stage, the developer preferred implementing all agile practices recommended for this stage. They stated that the complementary use of these practices would enhance identification of defects that might have escaped the prior stages, particularly functional defects. Similarly, at the software integration and integration testing stage, the

developer preferred a selection of PP, integration testing, refactoring, onsite customer/proxy. They affirmed that these practices would have addressed integration defects in the project. At both the software system testing and acceptance testing stages, the developer selected PP, UAT, and onsite customer/proxy for addressing defects. They emphasised that the involvement of various stakeholders during these stages would have ensured the software met both functional requirements and user expectations. This suggests that they would have identified potential functional, interface, and performance defects that might have escaped the prior stages.

Upon completion of the validation process, the developer asserted that following the AV4AD model could have prevented defects that emerged when the software was initially showcased to the client, thereby avoiding rework. Additionally, the developer indicated that they would recommend the model's adoption in future projects to the development team and management.

5.3.4 Validation in Organisation B

5.3.4.1 Profile of Organisation B

Organisation B was established in 2013 and is headquartered in Ireland. Currently, Organisation B has a total of seven fulltime personnel, including the CEO, Chief Clinical Officer, Chief Technical Officer (CTO), Lead Software Engineer, Quality Assurance Testing (QAT) professional, and two junior engineers. Additionally, the organisation contracts one parttime specialist. The organisation currently specialises in outcome measurement software for various healthcare institutions including public hospitals, mental health facilities, and university hospitals. The software falls in Class I category according to the Regulation (EU) 2017/745 (EU 2023). The initial version of the software was developed by a subcontracted company outside Ireland. However, it could not meet functional expectations and ultimately failed due to many defects including requirements and coding defects. Consequently, a complete redevelopment was undertaken by Organisation B's inhouse team. Organisation B currently follows Azure DevOps to implement improvements to their software and to add new features requested by their clients. Information related to the methodology followed in the development of the version of the software that failed was not available as the organisation was not adequately involved in the development process. Additionally, the CEO mentioned that the specification for the software was not provided by the subcontracted company.

5.3.4.2 Validation Process in Organisation B

The validation in Organisation B involved the development team retrospectively walking-through the software's redevelopment using the AV4AD model to assess whether it could have addressed defects. The team included the researcher and three members (CEO, CTO, and Lead Software Engineer) from the organisation. The CEO had participated in semi-structured interviews and was therefore familiar with the AV4AD model. However, the other two members needed an explanation about the model, which the researcher provided prior to its application. Thereafter, the researcher only intervened to ask questions and provide clarifications when necessary.

Following the researcher's explanation and prior to the model's application, the CTO commended the AV4AD model in relation to ongoing software improvement tasks in the organisation. They stated, "*I think it still applies even if you are improving, where you are not really creating major features, but even on bug fixes. Still, this model applies, yeah*". This endorsement was corroborated by the CEO who emphasised the model's relevance, stating:

"*Yeah, it does. Certainly, there is huge relevance to us, particularly in terms of the bottom green bar, ISO 13485. That's certainly a standard we would aspire to. We are doing a lot of those practises, or at least I think we are right now. But hopefully we can bring that together at some stage to start to look at certifications like that. So, there is a lot of relevance here and it's a good exercise for us to go through*".

Throughout the process the team discussed and agreed on practices they would have used at various stages of the development process to address defects.

5.3.4.3 Results from Organisation B

At the requirements gathering stage, the team selected user story and onsite customer/proxy for implementation. They affirmed that these practices would have addressed requirements defects in the project. The team outlined a process in which requirements (complaints about defects or new feature requests) generated by clients (onsite customer/proxy) and submitted via Freshdesk – the organisation's customer support platform, were discussed collaboratively. Thereafter, user stories were developed in consultation with the specific client. The team affirmed that this process would have addressed requirements defects.

During the requirements specification stage, the team preferred implementing all practices recommended for addressing defects at this stage. They noted that the initial implementation lacked a documented requirements specification. The team agreed that applying these practices would have addressed defects arising from ambiguities in requirements, interface defects, defects related to functionality and achieved appropriate specification. They stated that at this stage, user stories would be broken down into individual tasks and corresponding test cases written. These tasks would then be assigned to individual developers for implementation.

At the architectural and detailed design stages, all practices recommended for addressing defects at this stage were selected for implementation. The team confirmed that an iterative cycle of designing, developing, evaluating, and refining would have addressed defects including design defects, integration defects and interface defects. They affirmed all practices were necessary for preventing architectural and design defects in their project.

During the software unit implementation stage, the team affirmed that all practices recommended at this stage could address defects in their project. However, they specifically selected PP, TDD, coding standards, code review, CI, unit testing, and refactoring for implementation, asserting that these could have addressed defects including functional and coding defects.

At the software unit testing stage, all practices were selected for addressing defects. They stated that at this stage the development team as well as the QAT person and their 'power customers' in the proxy role, could have been involved. They agreed that involvement of various stakeholders could have enhanced defect identification and resolution. Similarly, at the last three stages, the team agreed that all practices would be used to address defects.

At the end of the validation process, the team affirmed that the AV4AD model could have addressed defects that led to initial project failure, thereby supporting successful delivery of the project outcomes. This observation was made in relation to the project's initial implementation which had failed to meet their expectations and satisfaction. The team further asserted that following the AV4AD model would enable their organisation to develop software while remaining compliant with the IEC 62304 standard.

5.3.5 Validation in Organisation C

5.3.5.1 Profile of Organisation C

Organisation C, an SME, was established in 2020 and develops medical diagnostic devices for identifying rare and inherited metabolic diseases. Their software falls in Class I category according to the Regulation (EU) 2017/745 (EU 2023). The development of the current version of the software involved five personnel; two senior software engineers, and three AI specialists. The organisation did not follow any standard SDLC in developing the software. The senior software engineer involved in the validation of the AV4AD model described their development approach as largely trial-and-error or opportunistic, indicating the absence of a structured or planned development process.

5.3.5.2 Validation Process in Organisation C

The implementation process was conducted retrospectively involving the researcher and one senior software engineer from the organisation who was the lead in the development of the software's version. At the beginning of the implementation process, the software engineer provided the background to the project. They explained that the project originated from research conducted by a researcher at an Irish University to address a problem related to rare and hereditary metabolic diseases in Ireland. Data used in their system about hereditary and rare diseases was collected from the Human Phenotype Ontology (HPO) sources that provide standardised vocabulary of specific clinical signs, symptoms, and structural abnormalities associated with hereditary and rare diseases. The software engineer explained that potential clients or end-users were only engaged when the development had been completed. They further noted that numerous defects were encountered throughout the process which contributed to significant delays in delivering the final software. As mentioned earlier, no standard SDLC was followed during development.

The software engineer and the researcher subsequently examined the project's implementation using the AV4AD model, focusing on whether it would have addressed the defects identified in the original development process. At each stage of the model, the software engineer assessed whether all practices would have been used or if only a selection would have been used. Additionally, the software engineer was asked to determine whether each of the implemented practices at a stage addressed defects. They were further asked to determine whether among the implemented practices, there was a practice(s) that worked better than the others in

addressing defects at a particular stage. The software engineer also provided an overall assessment of the AV4AD model focusing on its effectiveness in addressing defects within the project and its potential adoption in their future projects.

5.3.5.3 Results from Organisation C

During the requirements gathering stage, the software engineer selected onsite customer or proxy and user story for addressing defects. They explained that these practices would have addressed defects of ambiguities in requirements encountered, which had contributed to delays in delivery of the project. Reflecting on their initial implementation, the software engineer noted that the absence of these practices resulted in repeated iterations and eventual abandonment of several requirements. They stated:

"*Those are two things we didn't have, and I think that really kind of delayed or it extended the time of the project because we kept iterating through different things and a lot of that stuff ultimately wasn't useful and we dropped. We should have had clinicians there from the very beginning who are our user base*".

At the requirements specification stage, the software engineer affirmed all practices for addressing defects. However, they preferred model storming, PP, and onsite customer or proxy in their project, noting that these would have prevented defects arising from ambiguities in requirements, and coding defects. Reflecting on their initial implementation, the software engineer emphasised the missed opportunity to utilise model storming and PP, stating:

"*It would have been better to really put effort into model storming and then also … have the paired programming and have them check each other's code. … It would have prevented defects, defects just being kind of bad code and elongated development time*".

They further remarked that "*having the onsite customer or proxy there would have reduced development time put into features that weren't ultimately useful*".

At the software architectural design stage, the software engineer selected all practices for addressing defects. Reflecting on the initial implementation, they emphasised that simple design, onsite customer, and model storming would have provided greater clarity to the development team, thereby preventing design defects, interface defects, coding defects and reducing ambiguities in requirements. They also stressed the importance of integration testing at this stage, stating that it would have prevented integration defects in their project.

During the software detailed design, the software engineer selected all the agile practices included for addressing defects. They explained that these practices would have addressed defects similar to those encountered in the previous stage. The software engineer further stated that using the practices included at this stage would have resulted in the development of simpler and more modular designs, thereby preventing design defects and ultimately reducing the risk of coding defects due to unnecessary design complexity.

At the software unit implementation stage, the software engineer endorsed all the agile practices for addressing defects. However, the particularly selected coding standards, code ownership, code review, continuous integration, small release, refactoring, and onsite customer. They explained that these practices would have addressed coding defects, performance defects, and interface defects in their project and "*would have made the development process so much easier*".

At the software unit testing stage, the software engineer selected all the agile practices for addressing defects. They explained that although basic unit testing had been performed during the initial implementation, applying all of these practices "*would have helped in resolving code breaks and all these kinds of technical issues that we faced*". Their reflection points to the presence of coding defects and accumulated technical debt which often manifests as testing debt and can be a huge problem during the unit testing stage.

During the software integration and integration testing stage, the software engineer selected all the agile practices for addressing defects. They specifically emphasised using onsite customer at this stage for identifying defects and commended its application throughout the earlier stages "*so it would have identified defects way earlier*". The software engineer further stated that the agile practices would have addressed integration and interface defects they encountered in their initial implementation. At the software system testing stage, the software engineer selected all agile practices for addressing defects. They stated that the agile practices "*would have really helped mitigate a lot of the issues we did face*", including interface and performance defects.

During the acceptance testing stage, the software engineer selected all the agile practices included for addressing defects. Reflecting on their initial implementation, they noted that these practices would have addressed interface defects, quality defects, and performance defects. The software engineer further emphasised the importance of onsite customer noting that it would

have provided the necessary feedback that would have enabled them to address quality defects. They stated:

"*We never really got that feedback. ... but the end user is the most important and we didn't have that. So, it would have been really helpful to have this to force us to get an end user to test and would have mitigated all these more kind of quality issues rather than maybe technical issues*".

At the end of the implementation process, the software engineer confirmed that using the AV4AD model would have prevented the defects encountered and enabled the organisation to deliver the project satisfactorily within the planned timeline, while being compliant with relevant standards like IEC62304. They stated:

"*I think we would have had a better sense of the product of our kind of everything, so our code base, our requirements, our end users wants. We would have had a better idea of how in alignment we are with standards. It's very clear to me how we would have prevented defects and identified defects*".

The software engineer further stated that they would recommend adopting the AV4AD model in future projects in the organisation.

5.3.6 Validation in Organisation D

5.3.6.1 Profile of Organisation D

Organisation D is an SME established in 2015 that develops cognitive deficit medical devices for assessing, monitoring, or compensating for impairments in brain function. The organisation currently has a total of five employees: the CEO who is a clinical expert, software developer/engineer, a mechanical engineer, a quality specialist who is also a software developer, and a regulatory specialist. Organisation D's product falls in Class IIa category according to the MDR (EU) 2017/745 (EU 2023) and is classified as Class II by the FDA. Class IIa or Class II medical devices are medium or moderate risk devices.

5.3.6.2 Validation Process in Organisation D

The implementation process involved the researcher and the quality specialist who participated in the initial implementation of the project. At the beginning, the quality specialist commended the AV4AD model that "*it definitely would address defects*". The quality specialist then

provided background on the project focusing on its initial implementation. They explained that the initial implementation did not involve potential clients and that the team encountered defects, including ambiguities in requirements, architectural defects, interface defects, performance defects, and design defects. After providing background on the project, implementation using the AV4AD model commenced. Throughout the process, the quality specialist compared the AV4AD implementation with their initial hybrid approach. At each stage they identified which agile practices could have been used and assessed whether they would have helped to address the defects previously faced.

5.3.6.3 Results from Organisation D

At the requirements gathering stage, the quality specialist commended all the agile practices included for addressing defects. However, they selected user story and use case as particularly effective for preventing defects in their project. The quality specialist noted that "*user story helps to reduce defects because you understand the issue*". This indicates that user story could have helped in preventing defects encountered in their initial implementation such as ambiguities in requirements. The quality specialist further noted that use case was the most commonly used practice in their development team for identifying system functions and determining critical functionalities for subsequent analysis. They emphasised that the application of use cases "*reduces defects*". This observation underscores the importance of use cases in preventing defects of omission in the early stages of the development lifecycle.

At the software requirements analysis stage, the quality specialist affirmed all the agile practices for addressing defects. However, they selected BDD and PP as particularly important for addressing defects in their project, noting that these would have addressed defects related to ambiguities in requirements and omission. They explained that among the practices included at this age, only BDD had been used during the initial implementation. The quality specialist further recommended the complementary application of continuous testing and PP, noting that this combination would prevent defects at this stage. In the AV4AD model, however, this approach is accomplished through the complementary application of TDD, PP, and onsite customer at this stage.

During the architectural design stage, the quality specialist confirmed all the agile practices for addressing defects. However, they selected integration testing and simple design for addressing defects in their project. They explained that these practices would have addressed

"*architectural defects for integration of components or communication*". At the software detailed design stage, the quality specialist explained that their initial implementation did not focus on this stage. However, they emphasised the importance of the stage and the agile practices included for addressing defects. The quality specialist specifically selected integration testing, stating, "*the integration testing is key there to make sure that everything works together. So that reduces the defects*".

At the software unit implementation stage, the quality specialist selected all the agile practices for addressing defects. They explained that these practices would have addressed coding defects, performance defects, interface defects, and design defects in their projects. During the software unit testing stage, the quality specialist selected continuous integration, unit testing, refactoring, and PP. They emphasised that the complementary application of these practices "*would be key*" at this stage. However, the quality specialist acknowledged that following the application of the agile practices in the previous stages, there would be fewer or no defects found at this stage.

At the software integration and integration testing stage, the quality specialist confirmed the agile practices for addressing defects. However, they particularly selected integration testing and unit testing and emphasised the complementary application of these practices for addressing defects in their project. They iterated that this stage only fewer defects would be found as a result of appropriate application of agile practices in the early stages.

During the software system testing and acceptance testing stages, the quality specialist selected UAT and onsite customer or proxy. The stated that defects at this stage would have already been addressed following the application of the agile practices in the previous stages. Therefore, their focus would be on whether the system works as intended, whether it meets the requirements, and whether all the risk controls are met. They confirmed that using the AV4AD model many defects would be identified and resolved in the earlier stages.

At the end of the implementation process, the quality specialist commended the AV4AD model and affirmed its relevance and effectiveness in addressing defects. They described the model as "*a huge opportunity for addressing defects*", emphasising its value in satisfactory delivery of development products. The quality specialist also highlighted the importance of standards incorporated into the AV4AD model, noting that their inclusion enhances its rigour and applicability in MDS development.

5.3.7 Discussion

The results of the desk-based implementation showed that the AV4AD model addresses defects in MDS development. Additionally, the implementation confirmed the importance of the model's flexibility to enable developers to choose whether to implement all practices or a selection at a stage. This flexibility ensures that agile practices are applied in alignment with project-specific needs. The findings also suggest that the AV4AD model could have addressed defects in the four organisations, thereby leading to successful and satisfactory projects delivery. The model's emphasis on early user involvement reinforces its potential to prevent defects at early stages. Furthermore, the positive feedback from the developer in Organisation A describing the model as '*this model has everything you need, covers all the important pieces*', suggests its comprehensiveness for MDS development. Additionally, recommendations by the CEO and CTO of Organisation B, further affirms the relevance of the AV4AD model for addressing defects in MDS development. Additionally, the affirmation by the quality specialist from organisation D describing the AV4AD model as "*a huge opportunity for addressing defects*" shows the relevance of the model in MDS development.

Overall, the findings show that the AV4AD model provides a more proactive approach to addressing defects in MDS development. The model's emphasis on early defect prevention, collaborative teamwork, and the flexibility in selecting and using practices at a stage, potentially contributes to the development of quality software that meets user needs and expectations. The insights gained from the implementation suggests that adopting the AV4AD model could improve software quality and reduce costly consequences, making it a suitable approach for MDS development projects. The confirmation from the four organisations indicates that the AV4AD model can be applied in both plan driven and agile development environments.

5.4 Summary

This chapter has presented the evaluation and validation of the AV4AD model. The evaluation was done through semi-structured interviews conducted with eight developers from different MDS organisations in Ireland. Following the evaluation, a retrospective desk-based implementation of the model was conducted within four MDS organisations in Ireland. The findings from both the evaluation and validation indicate that the AV4AD model can address defects in MDS development. All eight interviewees affirmed that the model could address defects in MDS development while ensuring adherence to regulatory requirements.

Participants in the validation process affirmed that the model could have prevented defects that were identified in the projects' initial implementations. They emphasised the significance of agile practices in the early development stages to prevent defects. Furthermore, the findings support the importance of the flexibility the AV4AD model provides, enabling developers to use all or a selection of practices to address defects at a stge.

Overall, both the evaluation and validation results indicate that the AV4AD model provides a proactive approach to addressing defects in MDS development. Additionally, the results indicate that adopting this model could improve software quality and reduce costly consequences, thereby making it a suitable approach for MDS development projects. Moreover, findings indicate that the AV4AD model is suitable for both plan driven and agile development environments. The findings also suggest that organisations following the AV4AD model would develop software while remaining compliant with standards such as IEC62304.

6 Summary and Conclusions

6.1 Introduction

This thesis investigates the problem of defects in MDS development and presents the Agile V-Model for Addressing Defects (AV4AD) model to address these defects. This concluding chapter first presents a summary of the research conducted, followed by revisiting of the research questions and objectives. The chapter then outlines the contributions made by this research and discusses its limitations. Finally, the chapter identifies opportunities for further research and presents the overall conclusion.

6.2 Summary

This research was motivated by the problem of defects in MDS. The problem was identified through an initial exploratory literature review in the medical device domain. An extensive literature review was then performed to investigate the impact of MDS in healthcare and its development processes. The findings of the literature review presented in Chapter 2 revealed that defects in MDS are a growing concern in healthcare, causing injuries and death of patients. These defects are among the leading causes of numerous medical devices recalls in both the EU and the US. Medical device-related injuries, deaths, and recalls due to defects in MDS can lead to financial and reputational consequences for MDS manufacturers. The literature review also revealed that MDS is typically developed following plan-driven approaches due to their ability to produce deliverables for regulatory compliance. Among the various plan-driven approaches, the V-Model is the dominant approach, particularly because of its inherent verification and validation activities that are mandated by regulatory standards such as IEC62304.

To provide MDS organisations with a suitable solution for addressing defects, it was important to investigate the techniques currently utilised to address defects in MDS development. The review revealed that MDS organisations primarily utilise software testing to identify defects specifically during coding and testing stages of the software development lifecycle. Various software testing techniques such as taxonomy-based testing using generic taxonomies like IBM's Orthogonal Defect Classification (ODC) and HP's Defect Origins, Types, and Modes have been employed. Additionally, static analysis has been used for identifying defects in MDS.

Given the criticality of MDS, focusing exclusively on defect identification is inadequate and increases the likelihood of defects slipping through to the final software product. As discussed in Chapter 1, this issue is exacerbated by the difficulty in finding and fixing defects in the later stages of the development lifecycle. Ultimately, this can increase the risk of software failure, potentially causing catastrophic events (Wong et al. 2017). Moreover, identifying defects late in the software development life cycle can lead to costly rework.

Based on these findings, a solution was proposed to utilise agile practices to address defects throughout the MDS development lifecycle. Consequently, a review of the literature was conducted within the MDS domain to examine the adoption of agile software development and whether agile practices were specifically used for addressing defects. This review revealed that agile practices are integrated with plan-driven approaches for purposes such as accommodating changing requirements. Notably, only one study utilised onsite customer or user involvement specifically for addressing defects. The review then shifted focus to the general software domain, examining various agile methodologies to identify agile practices that could address defects. Through this process, 84 agile practices were initially selected which underwent inclusion and exclusion criteria resulting in 25 practices. The 25 agile practices were individually reviewed and analysed for their suitability to address defects in MDS. Ultimately, 17 agile practices were identified for inclusion in the proposed solution.

To complement the findings from the literature review, a survey was conducted with MDS developers in Ireland. This survey aimed to gather first-hand information about the agile practices currently used in MDS development and to determine whether any were used for addressing defects. The survey findings aligned with literature review findings, indicating an emphasis on defect identification with agile practices such as coding standards commonly used during coding and testing stages.

Having identified the suitable agile practices, the next stage of the research focused on determining the foundational SDLC to integrate the practices. Based on the literature review and survey findings, the V-Model emerged as the most appropriate foundation for the proposed solution. Consequently, seventeen agile practices were integrated into the V-Model forming the AV4AD model. The integration of these practices was based on insights gained through the literature review, analysis and survey conducted in this research. The AV4AD model was evaluated through semi-structured interviews with eight developers from different MDS organisations and was validated through its desk-based implementation within four MDS

organisations in Ireland. The data collected through semi-structured interviews were analysed and relevant feedback was incorporated into the model before its implementation. The findings from the evaluation and validation indicate that the AV4AD model provides a proactive approach to addressing defects, which reduces the risk of defects in the final release of MDS. This indicates that the model may be well-suited for MDS development projects, particularly due to its potential to minimise risks of patient injuries, deaths, financial loss, and reputational damage for MDS organisations.

6.3 Research Questions Revisited

This research aimed to address defects in MDS development by using agile practices to prevent defects in the early stages of the development lifecycle and identify those that may occur in later stages. Consequently, the main contribution of this research is the development of the AV4AD model. The research was guided by research questions (RQs) and research objectives (ROs). The research questions answered in this study are as follows:

RQ1: *Which agile practices can address software defects?*

RQ2: *Which agile practices are used to address defects in MDS development?*

RQ3: *How can agile practices be integrated in MDS development to address defects?*

These research questions were addressed through the following six research objectives (ROs):

RO1: *To identify agile practices that have been used to address software defects in the General Software Domain (GSD).*

RO2: *To identify which agile practices have been used in MDS development.*

RO3: *To identify agile practices that have been used for addressing defects in MDS development.*

RO4: *To investigate which agile practices used for addressing defects in GSD can also be used to address defects in MDS development.*

RO5: *To develop and evaluate a model that incorporates agile practices to address defects throughout the MDS development lifecycle.*

RO6: *To validate the model.*

The subsequent subsections detail how each research objective was fulfilled.

6.3.1 Research Objective 1

The first objective was *to identify agile practices that have been used to address software defects in the GSD.* This objective was addressed through a literature review focusing on ASD in the GSD.

Firstly, agile methods and their practices were reviewed. In total 84 agile practices were reviewed from various agile methods and underwent inclusion and exclusion criteria to determine those that could potentially address defects. This process resulted in a total of 25 agile practices that were then individually reviewed to ascertain their potential to address defects.

Secondly, a further review was conducted on each of the twenty-five agile practices in a total of 667 papers to gather evidence of their specific use for addressing defects and to gain insights into their application. The results showed that the majority of studies reported significant reduction in defects as a result of the agile practice(s) utilised. It was found that 15 of the 25 agile practices have been specifically utilised to address defects in the GSD. Based on the findings of this part of the research, 17 of the 25 agile practices have been recommended for addressing defects in MDS development.

6.3.2 Research Objective 2

The second objective was *to identify which agile practices have been used in MDS development*.

To address this objective, a literature review and survey were conducted focusing on the adoption of agile methodologies and practices in MDS development. Out of 117 reviewed papers, only 53 discussed the use of agile practices in MDS development. Among these, 34 were relevant to this study as they clearly indicated the specific agile practices used. Findings revealed that agile methodologies are not exclusively used in MDS development, particularly due to stringent regulatory requirements. These regulations demand extensive documentation and traceability which can make it difficult to fully embrace the flexibility and iterative nature of agile methodologies. Consequently, various agile practices are commonly integrated with the V-Model to address specific problems such as accommodating changing requirements. This integration has resulted in various hybrid models as discussed in Chapter 2. One such example

is the AV-Model which was designed to address the problem of accommodating changing requirements. Practices in this model include onsite customer, use case, user story, allocate timeboxing, sprint backlog, task board, burn-down chart, daily stand-up meeting, sprint retrospective, test first development, PP, collocated teams, and self-organizing teams.

The survey was distributed to 65 MDS organisations in Ireland and 15 responses were received. The results aligned with literature review findings particularly regarding the use of hybrid approaches. Additionally, respondents reported the exclusive use of agile methodologies. Furthermore, the survey showed that the most commonly used agile practices were coding standards, code review, unit testing, and integration testing – regardless of whether they were used in plan driven or exclusive agile approach.

6.3.3 Research Objective 3

The third objective was *to identify agile practices that have been used for addressing defects in MDS development.*

This objective was addressed in part, through a further examination of the integration of agile practices with plan-driven approaches as revealed in Objective 2. It was found that agile practices were not specifically used for addressing defects but rather other purposes such as accommodating changing requirements. Out of the 34 reviewed studies, only one used an agile practice specifically for addressing defects. This study used onsite customer or active user involvement and reported a significant reduction in defects. Additionally, this objective was further addressed through a survey of MDS organisations in Ireland. The survey findings indicated the use of agile practices for addressing defects commonly employed in the later stages of development, specifically during coding and testing stages. These findings aligned with the literature review findings which indicated emphasis on defect identification in MDS development. Among the various agile practices used for addressing defects, code review, unit testing, integration testing, coding standards, and user story were the most commonly used.

6.3.4 Research Objective 4

The fourth objective was *to investigate which agile practices used for addressing defects in the GSD can also be used to address defects in MDS development*.

Upon addressing the previous three objectives, it became apparent that the most suitable approach to MDS development must prioritise both defect prevention and defect identification.

It was determined that such a solution should incorporate agile practices to address defects throughout the MDS development lifecycle by preventing defects in the early stages and identifying any in later stages. Consequently, this objective was addressed through analysis of the 25 agile practices identified and reviewed in Objective 1, to determine their potential applicability to address defects in MDS development. As a result of this analysis, a total of 17 agile practices were identified for inclusion in the proposed solution. These practices include PP, TDD, CI, unit testing, refactoring, integration testing, onsite customer, code review, UAT, code ownership, coding standards, small releases, user story, simple design, use case, BDD, and model storming. Practices that have not been included are product backlog, sprint backlog, daily stand-up meeting, planning game, sprint, sprint planning meeting, sprint review meeting, retrospective.

6.3.5 Research Objective 5

The fifth objective was *to develop and evaluate a model that incorporates agile practices to address defects throughout the MDS development lifecycle*.

Having identified agile practices that address defects through the literature review, analysis, and survey, and having gained insights into their application for addressing defects, Objective 5 was then addressed. This was achieved through the development of the AV4AD model by integrating the 17 agile practices with the V-Model. The development of the AV4AD model is covered in Chapter 4. The V-Model was selected based on its dominant use for MDS development and its capability to produce deliverables for regulatory compliance as revealed in the literature presented in Chapter 2. The AV4AD model was evaluated through semi-structured interviews conducted with eight MDS developers from different MDS organisations in Ireland. The process of the semi-structured interviews is detailed in Chapter 3. The evaluation results indicate that the AV4AD model can help MDS organisations in proactively addressing defects while ensuring regulatory compliance.

Unlike existing models, the AV4AD model enables MDS organisations to address defects throughout the MDS development lifecycle by using agile practices that prevent defects in the early stages and those that identify defects in the later stages. This approach significantly reduces the risk of defects slipping to the final release, thereby avoiding costly consequences for MDS organisations and preventing patient injuries and deaths.

6.3.6 Research Objective 6

The sixth objective was *to validate the model*.

This objective aimed to assess the effectiveness of the AV4AD model in addressing defects in MDS development. To address this objective, the AV4AD model was retrospectively implemented in four MDS organisations in Ireland. The validation process is detailed in Chapter 5. The validation findings indicate that the AV4AD model offers a proactive approach to addressing defects and is suitable for MDS development in both plan-driven and agile environments. The findings further show that the AV4AD model can prevent various types of defects including those related to requirements, design, interfaces, integration, and functionality. Moreover, the findings suggest that the model's implementation could lead to increased MDS project successful and satisfactory delivery.

6.4 Research Contribution

This research makes a number of contributions to the domain of medical device software as well as to the broader field of software engineering. The contributions are organised into the following areas; each elaborated upon in the subsequent subsections:

- Literature review
- MDS organisations Survey
- Identification of agile practices for addressing defects
- Model development
- Conference papers

6.4.1 Contribution through Literature Review

The literature review explored and addressed various aspects related to MDS development. It revealed the critical problem of defects in MDS that lead to patient injuries, deaths, and recalls of numerous medical devices. These incidents can result in financial losses and reputational damage for MDS organisations. The review examined existing techniques for addressing these defects, revealing that MDS organisations rely heavily on defect identification mainly through software testing conducted after development is completed. This reactive approach increases the likelihood of defects escaping to the final software release. The challenge is exacerbated by the difficulty to find and fix defects in the later stages of software development lifecycle.

To effectively address defects in MDS development, a proactive approach that priorities both defect prevention and identification is necessary.

The review also examined various SDLC models and ASD methodologies, outlining their respective benefits and shortcomings. As discussed in Chapter 4, through inclusion and exclusion, the review identified 25 agile practices that could potentially address defects. Through an analysis of 667 papers detailed in Chapter 4, the list was reduced to 17 agile practices that can address defects in MDS development. These practices are important for MDS organisations aiming to leverage agile practices specifically for addressing defects. Additionally, the research provides guidance about how each of these practices can be effectively applied to address defects. Overall, the literature review presented in this research provides important insights and holds significant value for both the MDS domain and the wider software engineering community. By leveraging these findings, stakeholders can refine their development processes and contribute to the successful implementation of projects.

6.4.2 Contribution through MDS Organisations Survey

The literature review revealed a scarcity of empirical evidence regarding the use of agile practices to address defects in MDS development. This lack of evidence may discourage MDS organisations from adopting agile practices in their development processes due to concerns about potential failures that could jeopardise the already obtained regulatory approvals. However, the survey conducted in this research provides evidence of using agile practices to address defects in MDS development. Although only fifteen organisations participated in the survey, the insights gathered are valuable and can serve as reference for other MDS organisations aiming to apply agile practices to address defects. While the survey results indicate an emphasis on defect identification, other organisations can build upon these findings and effectively address defects.

6.4.3 Model Development

Research indicates that defects that arise during software use are often left undetected during the development lifecycle of the software (Gupta, and Sahu 2024). Given the safety-critical nature of MDS, it is important to utilise techniques that enable MDS organisations to address defects proactively throughout the development lifecycle. This research proposes the use of agile practices to prevent defects in the early stages of MDS development lifecycle and identify defects that may appear in the later stages. By integrating agile practices that address defects

throughout the development lifecycle, this approach enhances the ability to reduce the likelihood of defects in the final release of the software.

A key contribution of this research is the development of the AV4AD model – a hybrid SDLC model which integrates the 17 agile practices that address defects with the V-Model. This model is designed to enable MDS organisations to address defects at every stage of the development lifecycle while ensuring compliance with regulatory standards such as IEC62304. The AV4AD model addresses defects via two approaches: defect prevention in the early stages of the development lifecycle and defect identification in the later stages, by using agile practices.

The AV4AD model was evaluated through semi-structured interviews with eight developers from different MDS organisations, followed by validation through its retrospective desk-based implementation in four MDS organisations in Ireland. The evaluation results confirmed that the model is well-suited for addressing defects in MDS development. The validation results indicated that implementing the AV4AD model addresses defects and could have addressed defects observed in the four organisations. This would have led to more successful project outcomes, particularly in organisation B where the initial implementation had failed to meet expectations. Overall, the evaluation and validation findings support the AV4AD model as an effective solution for addressing defects in MDS development, while ensuring compliance with regulatory requirements.

6.4.4 Contribution through Conference Papers

This research also makes contribution through three peer-reviewed publications in international conference proceedings. The first paper titled "*Identifying Agile Practices for Addressing Defects in Medical Device Software Development*", was presented at the EuroSPI 2023 conference. The second paper, "*Using Agile Practices to Address Defects in Medical Device Software Development: A survey of Medical Device Software Development Organisations in Ireland*", was presented at the 2024 International Conference on Computer, Software and Modeling (ICCSM) conference. The third paper titled "*An Agile-based Framework for Addressing Defects in Medical Device Software Development*", was presented at the EuroSPI 2024 conference.

The published findings are relevant to the MDS domain as well as the broader software engineering community. These findings can inform stakeholders aiming to refine their

processes to address defects, thereby increasing the likelihood of projects success. Moreover, the published work lays a strong foundation for potential further research and can support future research in related domains. The relevant feedback obtained from these conferences was important in this research. Notably, the beta version of the AV4AD model was presented at both the ICCSM and EuroSPI 2024 conferences, and feedback gathered was instrumental in shaping the AV4AD model before its implementation. Importantly, this research does not advocate for the abandonment of existing software testing techniques but rather promotes and provides a proactive approach to defect management that priorities both defect prevention and identification. This approach significantly reduces the likelihood of defects in the final software release, ultimately preventing catastrophic consequences.

6.5 Research Limitations

There are several limitations that should be acknowledged in this research. MDS organisations that were involved in this research are from Ireland. While this provided depth and valuable insights, a broader scope including other EU countries or organisations across the globe would have increased the generalisability of findings. However, in any research, defining a clear scope is important to ensure efficient use of resources. Due to time constraints, it was not possible to include a wider pool of organisations for data collection. Additionally, as with any research, nonresponse from potential participants due to time limitations or busy schedules, further reduced the range of data collection. It is therefore possible that the findings presented here may differ if either a larger or a different sample was used. Nonetheless, the important consideration in this research was to prioritise the collection of rich data which enabled gaining deeper insights into how MDS is developed, whether agile practices were used for addressing defects and how they were applied for this purpose.

Although the research plan included evaluating and validating the AV4AD model across several MDS organisations, nonresponse constrained these activities. Consequently, validation was conducted in four MDS organisations, all classified as SMEs. The AV4AD model, founded on the V-Model – widely used in MDS development and other safety-critical domains, has undergone evaluation and validation, and has been published in international conferences. However, broader implementation across a larger number of organisations and different MDS classifications would strengthen claims of generalisability. Despite these limitations, the evaluation and validation of the AV4AD model yielded valuable findings. These findings, in alignment with the conducted literature review and survey offer valuable contributions to the

MDS domain and the wider software engineering community. Although the AV4AD model was only implemented within SMEs and applied to MDS classified as Class I and Class II or IIa during this study, it is designed for use by any MDS organisation seeking to address defects, irrespective of organisational size and MDS classification or associated risk level. The absence of validation from large organisations, as well as the high-risk MDS classification, resulted from lack of responses from these organisations.

6.6 Further Research

This study developed and evaluated the AV4AD model which was implemented in four SMEs involved in MDS development. Although the implementation was only conducted in SMEs and on MDS Classified as Class I and Class II or IIa, the AV4AD model is designed for use by any MDS organisation seeking to address defects, regardless of organisational size, MDS classification, or associated risk level. To make stronger claims of generalisability, future research could extend the application of the AV4AD model to a larger number of MDS organisations including SMEs and large organisations, developing software of classifications A, B, and C, and situated in different geographical regions. A broader implementation of the AV4AD model would provide the opportunity to gather further perspectives on the effectiveness of the model and identify further opportunities for improvements. Additionally, future studies could explore the use of this model in contexts other than MDS that encounter similar problem of defects, such as automotive and avionics domains, as well as non-critical software domain.

Another area of future research involves conducting a survey targeting MDS organisations in different geographical locations across the globe to gather data on the use of agile practices for addressing defects in MDS development. Although the organisations involved in this research have international visibility, they may not be present in all countries. Given the growth of the MedTech industry over the years, collecting data from various organisations across the globe could provide better insights into the current development approaches and whether agile practices are used for addressing defects and how they are applied. Furthermore, future studies could broaden the scope of this survey to include organisations in other safety-critical domains and non-critical domains across different geographical regions. Comparative analysis across these domains would enable gathering broader insights in relation to application of agile practices for addressing defects. Such research could inform further improvements in using

agile practices to address defects within the MDS domain and contribute to best practices across related domains.

6.7 Conclusions

This project has investigated the problem of defects in MDS development and their consequences for both patients and MDS organisations. Defects in MDS are a growing concern for the healthcare domain, causing patient injuries, deaths, and numerous recalls of medical devices in both the US and EU. These incidents ultimately lead to financial losses and reputational damage for MDS organisations. Existing techniques for addressing defects in MDS development tend to be reactive, primarily emphasising defect identification when development has been completed. This reactive approach increases the risk of defects slipping to the final release, potentially leading to catastrophes.

Given the safety-critical nature of MDS, this research advocates for a proactive approach that prioritises both defect prevention and identification. This approach has been actualised through development of the AV4AD model which is designed to assist any MDS organisation seeking to address defects, regardless of organisational size, MDS classification, or associated risk level. The AV4AD model encompasses flexibility and adaptability, enabling both defect prevention and identification throughout the development lifecycle. By leveraging agile practices, the AV4AD model facilitates early defect detection and correction. This makes it a valuable tool for MDS organisations aiming to improve software quality and reduce the risk of defects in later development stages and the final release – where they can result in costly consequences.

The evaluation and validation results confirm the suitability of the AV4AD model for MDS development and addressing defects. In conclusion, the AV4AD model enhances the ability to minimise defect occurrence in the final software release and is adaptable for both plan driven and agile environments.

7 References

A Dongo, T., H. Reed, A. and T. O'Hara, M. (2016). Exploring Pair Programming Benefits for MIS Majors. *Journal of Information Technology Education: Innovations in Practice*, 15, pp.223–239.

AAMI. (2012). AAMI TIR45: 2012; Guidance on the use of agile practices in the development of medical device software., 2012. Available from: https://webstore.ansi.org/standards/aami/aamitir452012r2018.

AAMI. (2023). AAMI TIR45: 2023 Technical Information Report Guidance on the use of Agile Practices in the Development of Medical Device Software., 15 March 2023. Available from: https://array.aami.org/doi/10.2345/9781570208683.

AAMI/IEC. (2016). *AAMI/IEC TIR80002-3:2016 Medical Device Software - PART 3: PROCESS RE* [online]. Available from: https://shop.standards.ie/en-ie/standards/ [accessed 28 January 2025].

AAMI/IEC. (2015). BS EN 62304-2006+A1:2015 Medical Device Software - Software life-cycle processes., 2015.

Abdelaziz, A.A., El-Tahir, Y. and Osman, R. (2015). Adaptive Software Development for developing safety critical software. In: *2015 International Conference on Computing, Control, Networking, Electronics and Embedded Systems Engineering (ICCNEEE)*. Khartoum, Sudan: IEEE, pp.41–46. Available from: http://ieeexplore.ieee.org/document/7381425/ [accessed 26 December 2021].

Abou Khalil, Z., Constantinou, E., Mens, T., Duchien, L. and Quinton, C. (2019). A Longitudinal Analysis of Bug Handling Across Eclipse Releases. In: *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME). Cleveland, OH, USA: IEEE, pp.1–12. Available from: https://ieeexplore.ieee.org/document/8919074/ [accessed 16 February 2023].

Abou Khalil, Z., Constantinou, E., Mens, T. and Duchien, L. (2021). On the impact of release policies on bug handling activity: A case study of Eclipse. *Journal of Systems and Software*, 173, p.110882.

Abraham, J. (2015). Verification and Validation Spanning Models to Code., 2015. Available from: https://www.mathworks.com/content/dam/mathworks/tag-team/Objects/v/82901_82139_AIAA_2015_Paper_VnV_models_to_code-FINAL.pdf [accessed 8 March 2023].

Abrahamsson, P., Salo, O. and Ronkainen, J. (2002). Agile Software Development Methods: Review and Analysis., 2002, p.112.

Adams, W.C. (2015). Conducting Semi-Structured Interviews. In: Newcomer, K. E., Hatry, H. P., and Wholey, J. S., eds *Handbook of Practical Program Evaluation*. 1st edn Wiley, pp.492–505. Available from: https://onlinelibrary.wiley.com/doi/10.1002/9781119171386.ch19 [accessed 23 April 2024].

Addagarrala, K.C. and Kinnicutt, P. (2018). Safety critical software ground rules. *International Journal of Engineering & Technology*, 7(2.28), pp.344–350.

Adkins, B. and Smith, R. (2016). Agile Team Roles Product Owner & ScrumMaster., January 2016. Available from: https://www.agilealliance.org/Fwp-content/uploads/2016/01/The-Dynamic-Duo-PO-and-SM-Partnership-Agile-2012-bkadkins-ricksmith.pdf.

Adom, D., Yeboah, A. and Ankrah, A.K. (2016). Constructivism Philosophical Paradigm: Implication for Research, Teaching and Learning., 4(10), pp.1–9.

Ahamed, M.A. (2007). Backlog Clearance Planning Index – A proactive approach to Backlog Management. In: *International Project Management Leadership Conference (PML 2007)*. International Project Management Leadership Conference (PML 2007). p.9.

Ahmed Khalid, T. and Yeoh, E.-T. (2021). Enhancing software development cost control by forecasting the cost of rework: preliminary study. *Indonesian Journal of Electrical Engineering and Computer Science*, 21(1), p.524.

Aithal, A. and Aithal, P.S. (2020). Development and Validation of Survey Questionnaire & Experimental Data – A Systematic Review-based Statistical Approach., 2 November 2020. Available from: https://zenodo.org/record/4179499 [accessed 2 November 2024].

Akbari, Z., Khoshnevis, S. and Mohsenzadeh, M. (2017). A Method for Prioritizing Integration Testing in Software Product Lines Based on Feature Model. *International Journal of Software Engineering and Knowledge Engineering*, 27(04), pp.575–600.

Alemzadeh, H., Iyer, R.K., Kalbarczyk, Z. and Raman, J. (2013a). Analysis of Safety-Critical Computer Failures in Medical Devices. *IEEE Security Privacy*, 11(4), pp.14–26.

Alemzadeh, H., Iyer, R.K., Kalbarczyk, Z. and Raman, J. (2013b). Analysis of Safety-Critical Computer Failures in Medical Devices. *IEEE Security & Privacy*, 11(4), pp.14–26.

Ali, H. and Birley, S. (1999). Integrating deductive and inductive approaches in a study of new ventures and customer perceived risk. *Qualitative Market Research: An International Journal*, 2(2), pp.103–110.

Ali, Z. (2019). Behaviour-Driven Development as an Error-Reduction Practice for Mobile Application Testing., 31 March 2019. Available from: https://zenodo.org/record/3234110 [accessed 1 February 2023].

AlOmar, E.A., AlRubaye, H., Mkaouer, M.W., Ouni, A. and Kessentini, M. (2021). Refactoring practices in the context of modern code review: an industrial case study at Xerox. In: *Proceedings of the 43rd International Conference on Software Engineering: Software Engineering in Practice*. ICSE-SEIP '21. Virtual Event, Spain: IEEE Press, pp.348–357. Available from: https://doi.org/10.1109/ICSE-SEIP52600.2021.00044 [accessed 12 September 2022].

Alotaibi, Y.K. and Federico, F. (2017). The impact of health information technology on patient safety. *Saudi Medical Journal*, 38(12), pp.1173–1180.

Alsaqqa, S., Sawalha, S. and Abdel-Nabi, H. (2020). Agile Software Development: Methodologies and Trends. *International Journal of Interactive Mobile Technologies (iJIM)*, 14(11), pp.246–270.

Alshazly, A.A., Elfatatry, A.M. and Abougabal, M.S. (2014). Detecting defects in software requirements specification. *Alexandria Engineering Journal*, 53(3), pp.513–527.

Alshenqeeti, H. (2014). Interviewing as a Data Collection Method: A Critical Review. *English Linguistics Research*, 3(1), p.p39.

Alvertis, I., Koussouris, S., Papaspyros, D., Arvanitakis, E., Mouzakitis, S., Franken, S., Kolvenbach, S. and Prinz, W. (2016). User Involvement in Software Development Processes. *Procedia Computer Science*, 97, pp.73–83.

Ambler, S.W. (2007). Agile Model Driven Development (AMDD)., 2007.

Ambler, S.W. (2008). Agile Software Development at Scale. In: Meyer, B., Nawrocki, J. R., and Walter, B., eds *Balancing Agility and Formalism in Software Engineering*. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, pp.1–12. Available from: http://link.springer.com/10.1007/978-3-540-85279-7_1 [accessed 4 February 2025].

Ambler, S.W. and Lines, M. (2013). Disciplined Agile Delivery., December 2013. Available from: https://www.oreilly.com/library/view/disciplined-agile-delivery/9780132810098/.

Amit, I. and Feitelson, D.G. (2019). Which Refactoring Reduces Bug Rate? In: *Proceedings of the Fifteenth International Conference on Predictive Models and Data Analytics in Software Engineering*. PROMISE'19: The Fifteenth International Conference on Predictive Models and Data Analytics in Software Engineering. Recife Brazil: ACM, pp.12–15. Available from: https://dl.acm.org/doi/10.1145/3345629.3345631 [accessed 12 August 2022].

Amrit, C. and Meijberg, Y. (2018). Effectiveness of Test-Driven Development and Continuous Integration: A Case Study. *IT Professional*, 20(1), pp.27–35.

Anand, A., Kaur, J., Singh, O. and H. Alhazmi, O. (2021). Optimal Sprint Length Determination for Agile-Based Software Development. *Computers, Materials & Continua*, 68(3), pp.3693–3712.

Andriyani, Y., Hoda, R. and Amor, R. (2017). Reflection in Agile Retrospectives. In: Baumeister, H., Lichter, H., and Riebisch, M., eds *Agile Processes in Software Engineering and Extreme Programming*. Lecture Notes in Business Information Processing. Cham: Springer International Publishing, pp.3 – 19. Available from: http://link.springer.com/10.100 7/978-3-319-57633-6_1 [accessed 17 October 2022].

ANSI-AAMI. (2018). *AAMI TIR45:2012 (R2018) Guidance on the use of Agile Practices in the development of medical device software*.

ANSI/AAMI. (2018). *ANSI/AAMI SW91: 2018 Classification of Defects in Health Software*.

Antinyan, V., Derehag, J., Sandberg, A. and Staron, M. (2018). Mythical Unit Test Coverage. *IEEE Software*, 35(3), pp.73–79.

Anwar, A. (2014). Rational Unified Process. *International Journal of Software Engineering*, 2014.

Avedian, A. (2014). Survey Design., 15 October 2014.

Ayhan, H.Ö. (2011). Non-probability Sampling Survey Methods. In: Lovric, M., ed. *International Encyclopedia of Statistical Science*. Berlin, Heidelberg: Springer, pp.979–982. Available from: https://doi.org/10.1007/978-3-642-04898-2_41 [accessed 26 June 2023].

Azad, N. and Hyrynsalmi, S. (2023). DevOps critical success factors — A systematic literature review. *Information and Software Technology*, 157, p.107150.

Azevedo Santos, M. de, Souza Bermejo, P.H. de, Oliveira, M.S. de, Tonelli, A.O. and Seidel, E.J. (2013). Improving the Management of Cost and Scope in Software Projects Using Agile Practices. *International Journal of Computer Science and Information Technology*, 5(1), pp.47–64.

Bacchelli, A. and Bird, C. (2013). Expectations, outcomes, and challenges of modern code review. In: *2013 35th International Conference on Software Engineering (ICSE)*. 2013 35th International Conference on Software Engineering (ICSE). San Francisco, CA, USA: IEEE, pp.712–721. Available from: http://ieeexplore.ieee.org/document/6606617/ [accessed 22 August 2022].

Bacchelli, A., Ciancarini, P. and Rossi, D. (2008). On the Effectiveness of Manual and Automatic Unit Test Generation. In: *2008 The Third International Conference on Software Engineering Advances*. 2008 The Third International Conference on Software Engineering Advances. pp.252–257.

Badanahatti, A. and Pillutla, S. (2020). Interleaving Software Craftsmanship Practices in Medical Device Agile Development. In: *Proceedings of the 13th Innovations in Software Engineering Conference on Formerly known as India Software Engineering Conference*. ISEC 2020. New York, NY, USA: Association for Computing Machinery, pp.1–5. Available from: https://doi.org/10.1145/3385032.3385047 [accessed 13 March 2023].

Bagheri, A. and Hegedus, P. (2022). Is Refactoring Always a Good Egg? Exploring the Interconnection Between Bugs and Refactorings., 2022, p.5.

Bakhtiary, V., Gandomani, T.J. and Salajegheh, A. (2020). The effectiveness of test-driven development approach on software projects: A multi-case study. *Bulletin of Electrical Engineering and Informatics*, 9(5), pp.2030–2037.

Balaji, S. (2012). Waterfall vs V-Model vs Agile: A Comperative Study on SDLC. *Vol.*, 2012.

Balijepally, Mahapatra, Nerur, and Price. (2009). Are Two Heads Better than One for Software Development? The Productivity Paradox of Pair Programming. *MIS Quarterly*, 33(1), p.91.

Baljé, J., Carter, A. and Velthuijsen, H. (2015). Agile Development as a Change Management Approach in Healthcare Innovation Projects., October 2015, p.16.

Barros-Justo, J.L., Benitti, F.B.V. and Tiwari, S. (2019). The impact of Use Cases in real-world software development projects: A systematic mapping study. *Computer Standards & Interfaces*, 66, p.103362.

Basak, S. and Shazzad Hosain, Md. (2014). Software Testing Process Model from Requirement Analysis to Maintenance. *International Journal of Computer Applications*, 107(11), pp.14–22.

Bassil, Y. (2012). A Simulation Model for the Waterfall Software Development Life Cycle. *International Journal of Engineering*, 2(5).

Baumeister, R.F. and Leary, M.R. (1997). Writing Narrative Literature Reviews., 1997.

Bavota, G., De Lucia, A., Di Penta, M., Oliveto, R. and Palomba, F. (2015). An experimental investigation on the innate relationship between quality and refactoring. *Journal of Systems and Software*, 107, pp.1–14.

Bavota, G., De Carluccio, B., De Lucia, A., Di Penta, M., Oliveto, R. and Strollo, O. (2012). When Does a Refactoring Induce Bugs? An Empirical Study. In: *2012 IEEE 12th International Working Conference on Source Code Analysis and Manipulation*. 2012 12th IEEE Working Conference on Source Code Analysis and Manipulation (SCAM). Riva del Garda, Italy: IEEE, pp.104–113. Available from: http://ieeexplore.ieee.org/document/6392107/ [accessed 28 July 2022].

Bavota, G. and Russo, B. (2015). Four eyes are better than two: On the impact of code reviews on software quality. In: *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME). Bremen, Germany: IEEE, pp.81 – 90. Available from: http://ieeexplore.ieee.org/document/7332454/ [accessed 25 August 2022].

Beck, K., Beedle, M., Bennekum, A. van, Cockburn, A., Cunningham, W., Fowler, M., Martin, R.C.M., Mellor, S., Thomas, D., Grenning, J., Highsmith, J., Hunt, A., Jeffries, J., Kern, J., Marick, B., Schwaber, K. and Sutherland, J. (2001). Agile-Manifesto-Color.pdf., 2001. Available from: https/www.agilealliance.org/wp-content/uploads/2017/10/Agile-Manifesto-Color.pdf [accessed 29 December 2024].

Beck, K. (2000). *extreme programming eXplained: embrace change*. Reading, MA: Addison-Wesley.

Beck, K. and Andres, C. (2004). *Extreme Programming Explained: Embrace Change*. 2nd edn Addison-Wesley Professional.

Beecham, S., Noll, J. and Richardson, I. (2014). Using Agile Practices to Solve Global Software Development Problems -- A Case Study. In: *2014 IEEE International Conference on Global Software Engineeering Workshops*. Shanghai, China: IEEE, pp.5–10. Available from: http://ieeexplore.ieee.org/document/6912078/ [accessed 6 April 2022].

Bekefi, B.F., Szabados, K. and Kovacs, A. (2019). A case study on the effects and limitations of refactoring. In: *2019 IEEE 15th International Scientific Conference on Informatics*. 2019 IEEE 15th International Scientific Conference on Informatics. Poprad, Slovakia: IEEE, pp.000213–000218. Available from: https://ieeexplore.ieee.org/document/9119321/ [accessed 25 July 2022].

Bekele, W.B. and Ago, F.Y. (2022). Sample Size for Interview in Qualitative Research in Social Sciences: A Guide to Novice Researchers. *Research in Educational Policy and Management*, 4(1), pp.42–50.

di Bella, E., Fronza, I., Phaphoom, N., Sillitti, A., Succi, G. and Vlasenko, J. (2013). Pair Programming and Software Defects--A Large, Industrial Case Study. *IEEE Transactions on Software Engineering*, 39(7), p.24.

Beller, M., Bacchelli, A., Zaidman, A. and Juergens, E. (2014). Modern code reviews in open-source projects: which problems do they fix? In: *Proceedings of the 11th Working Conference on Mining Software Repositories - MSR 2014*. the 11th Working Conference. Hyderabad, India: ACM Press, pp.202 – 211. Available from: http://dl.acm.org/citation.cfm?doid=2597073.2597082 [accessed 1 September 2022].

Benz, S. (2007). Combining test case generation for component and integration testing. In: *Proceedings of the 3rd international workshop on Advances in model-based testing*. ISSTA07: International Symposium on Software Testing and Analysis. London United Kingdom: ACM, pp.23–33. Available from: https://dl.acm.org/doi/10.1145/1291535.1291538 [accessed 31 May 2023].

Bhadauria, V., Mahapatra, R. and Nerur, S. (2020). Performance Outcomes of Test-Driven Development: An Experimental Investigation. *Journal of the Association for Information Systems*, 21, pp.1045–1071.

Bhat, T. and Nagappan, N. (2006). Evaluating the efficacy of test-driven development: industrial case studies. In: *Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering*. ISESE '06. New York, NY, USA: Association for Computing Machinery, pp.356 – 363. Available from: https://doi.org/10.1145/1159733.1159787 [accessed 9 June 2022].

Binamungu, L.P., Embury, S.M. and Konstantinou, N. (2020). Characterising the Quality of Behaviour Driven Development Specifications. In: Stray, V. et al., eds *Agile Processes in Software Engineering and Extreme Programming*. Lecture Notes in Business Information Processing. Cham: Springer International Publishing, pp.87–102.

Bird, C., Nagappan, N., Murphy, B., Gall, H. and Devanbu, P. (2010). An Analysis of the Effect of Code Ownership on Software Quality across Windows, Eclipse, and Firefox., 2010.

Bird, C., Nagappan, N., Murphy, B., Gall, H. and Devanbu, P. (2011). Don't touch my code! examining the effects of ownership on software quality. In: *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. ESEC/FSE'11: Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering. Szeged Hungary: ACM, pp.4–14. Available from: https://dl.acm.org/doi/10.1145/2025113.2025119 [accessed 13 February 2023].

Bliznakov, Z., Mitalas, G. and Pallikarakis, N. (2007). Analysis and Classification of Medical Device Recalls. In: Magjarevic, R. and Nagel, J. H., eds *World Congress on Medical Physics and Biomedical Engineering 2006*. IFMBE Proceedings. Berlin, Heidelberg: Springer Berlin Heidelberg, pp.3782–3785. Available from: http://link.springer.com/10.1007/978-3-540-36841-0_957 [accessed 5 March 2022].

Bliznakov, Z., Stavrianou, K. and Pallikarakis, N. (2014). Medical Devices Recalls Analysis Focusing on Software Failures during the Last Decade. In: Roa Romero, L. M., ed. *XIII Mediterranean Conference on Medical and Biological Engineering and Computing 2013*.

IFMBE Proceedings. Cham: Springer International Publishing, pp.1174–1177. Available from: https://link.springer.com/chapter/10.1007/978-3-319-00846-2_291.

Boddy, C.R. (2016). Sample size for qualitative research. *Qualitative Market Research: An International Journal*, 19(4), pp.426–432.

Boehm, B. and Basili, V.R. (2001). Software Defect Reduction Top 10 List. *Computer*, 34(1), pp.135–137.

Boehm, B.W. (1988). A spiral model of software development and enhancement. *Computer*, 21(5), pp.61–72.

Boogerd, C. and Moonen, L. (2008). Assessing the value of coding standards: An empirical study. In: *2008 IEEE International Conference on Software Maintenance*. 2008 IEEE International Conference on Software Maintenance (ICSM). Beijing, China: IEEE, pp.277–286. Available from: http://ieeexplore.ieee.org/document/4658076/ [accessed 21 March 2023].

Bottacci, E., Semi Rantanen, Romppanen, J., Noella Verschuren and Ahmed, F. (2022). DevOps Practices for Software Development and Consulting Firms a case for EfiCode., 2022. Available from: https://rgdoi.net/10.13140/RG.2.2.19777.89440 [accessed 5 July 2023].

Braga, A.M., Schwab, D.C. and Vannucci, A.L. (2015). The Use of Acceptance Test-Driven Development in the Construction of Cryptographic Software., 2015, p.6.

Braspenning, N.C.W.M., Van De Mortel-Fronczak, J.M. and Rooda, J.E. (2006). A Model-based Integration and Testing Method to Reduce System Development Effort. *Electronic Notes in Theoretical Computer Science*, 164(4), pp.13–28.

Braun, V., Clarke, V., Boulton, E., Davey, L. and McEvoy, C. (2021). The online survey as a qualitative research tool. *International Journal of Social Research Methodology*, 24(6), pp.641–654.

Brinkmann, S. (2023). *Qualitative Interviewing: Conversational Knowledge Through Research Interviews* [online]. 2nd edn Oxford University PressNew York. Available from: https://academic.oup.com/book/44447 [accessed 26 August 2024].

vom Brocke, J., Hevner, A. and Maedche, A. (2020). Introduction to Design Science Research. In: vom Brocke, J., Hevner, A., and Maedche, A., eds *Design Science Research. Cases*. Progress in IS. Cham: Springer International Publishing, pp.1–13. Available from: https://doi.org/10.1007/978-3-030-46781-4_1 [accessed 7 November 2022].

Brown, N. (2015). Epistemology. *Dr Nicole Brown* [online], 3 November 2015. Available from: https://www.nicole-brown.co.uk/epistemology/ [accessed 23 November 2022].

Bryant, S., Romero, P. and du Boulay, B. (2008). Pair programming and the mysterious role of the navigator. *International Journal of Human-Computer Studies*, 66(7), pp.519–529.

Buchan, J., Li, L. and MacDonell, S.G. (2011). Causal Factors, Benefits and Challenges of Test-Driven Development: Practitioner Perceptions. In: *2011 18th Asia-Pacific Software Engineering Conference*. 2011 18th Asia Pacific Software Engineering Conference. Ho Chi Minh, Vietnam: IEEE, pp.405 – 413. Available from: http://ieeexplore.ieee.org/document/6130648/ [accessed 2 May 2022].

Capiluppi, A., Boldyreff, C., Beecher, K. and Adams, P.J. (2009). Quality Factors and Coding Standards – a Comparison Between Open Source Forges. *Electronic Notes in Theoretical Computer Science*, 233, pp.89–103.

Carter, N., Bryant-Lukosius, D., DiCenso, A., Blythe, J. and Neville, A.J. (2014). The Use of Triangulation in Qualitative Research. *Oncology Nursing Forum*, 41(5), pp.545–547.

Cauevic, A., Punnekkat, S. and Sundmark, D. (2012). Quality of Testing in Test Driven Development. In: *2012 Eighth International Conference on the Quality of Information and Communications Technology*. 2012 Eighth International Conference on the Quality of Information and Communications Technology. pp.266–271.

Čaušević, A., Shukla, R. and Punnekkat, S. (2013). Industrial study on test driven development: Challenges and experience. In: *2013 1st International Workshop on Conducting Empirical Studies in Industry (CESI)*. 2013 1st International Workshop on Conducting Empirical Studies in Industry (CESI). pp.15–20.

Čaušević, A., Sundmark, D. and Punnekkat, S. (2012). Test case quality in test driven development: A study design and a pilot experiment. In: *16th International Conference on Evaluation & Assessment in Software Engineering (EASE 2012)*. 16th International Conference on Evaluation & Assessment in Software Engineering (EASE 2012). pp.223–227.

Cawley, O., Wang, X. and Richardson, I. (2010). Lean/Agile Software Development Methodologies in Regulated Environments – State of the Art. In: Abrahamsson, P. and Oza, N., eds *Lean Enterprise Software and Systems*. Lecture Notes in Business Information Processing. Berlin, Heidelberg: Springer Berlin Heidelberg, pp.31–36. Available from: http://link.springer.com/10.1007/978-3-642-16416-3_4 [accessed 28 October 2021].

Chakravarty, K. and Singh, J. (2022). Optimizing Defect Removal Efficiency by Defect Prediction using Machine Learning. In: *2022 OITS International Conference on Information Technology (OCIT)*. 2022 OITS International Conference on Information Technology (OCIT). Bhubaneswar, India: IEEE, pp.205 – 210. Available from: https://ieeexplore.ieee.org/document/10053742/ [accessed 4 March 2024].

Chong, J., Plummer, R., Leifer, L., Klemmer, S.R., Eris, O. and Toye, G. (2005). Pair Programming: When and Why it Works., June 2005, p.6.

Chowdhury, A.F. and Huda, M.N. (2011). Comparison between Adaptive Software Development and Feature Driven Development. In: *Proceedings of 2011 International Conference on Computer Science and Network Technology*. 2011 International Conference on Computer Science and Network Technology (ICCSNT). Harbin, China: IEEE, pp.363–367. Available from: http://ieeexplore.ieee.org/document/6181977/ [accessed 10 January 2025].

Chug, A. and Gupta, M. (2017). A quality enhancement through defect reduction using refactoring operation. In: *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI). Udupi: IEEE, pp.1869–1875. Available from: http://ieeexplore.ieee.org/document/8126117/ [accessed 26 July 2022].

Chukwuere, J.E. (2023). Exploring Literature Review Methodologies in Information Systems Research: A Comparative Study. *Education & Learning in Developing Nations*, 1(1), pp.74–82.

Cockburn, A. (2005). *Crystal Clear: A Human-powered Methodology for Small Teams*. Addison-Wesley.

Cohen, D., Lindvall, M. and Costa, P. (2004). An Introduction to Agile Methods. In: *Advances in Computers*. Elsevier, pp.1 – 66. Available from: https://linkinghub.elsevier.com/retrieve/pii/S0065245803620012 [accessed 4 January 2025].

Cohn, M. (2009). *User Stories Applied for Agile Software Development*. Boston, MA, USA: Addison-Wesley. Available from: www.wowebook.com.

Collard, R. (1999). Developing test cases from use cases. *Software Testing*, 1999.

Conboy, K. and Fitzgerald, B. (2010). Method and developer characteristics for effective agile method tailoring: A study of XP expert opinion. *ACM Transactions on Software Engineering and Methodology*, 20(1), p.2:1-2:30.

Connor, M.J., Tringale, K., Moiseenko, V., Marshall, D.C., Moore, K., Cervino, L., Atwood, T., Brown, D., Mundt, A.J., Pawlicki, T., Recht, A. and Hattangadi-Gluth, J.A. (2017). Medical Device Recalls in Radiation Oncology: Analysis of US Food and Drug Administration Data, 2002-2015. *International Journal of Radiation Oncology*Biology*Physics*, 98(2), pp.438–446.

Cook, D.A. (2019). Systematic and Nonsystematic Reviews: Choosing an Approach. In: Nestel, D. et al., eds *Healthcare Simulation Research: A Practical Guide*. Cham: Springer International Publishing, pp.55–60. Available from: https://doi.org/10.1007/978-3-030-26837-4_8 [accessed 5 June 2026].

Cordeiro, L., Barreto, R., Barcelos, R., Oliveira, M., Lucena, V. and Maciel, P. (2007). Agile Development Methodology for Embedded Systems: A Platform-Based Design Approach. In: *14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)*. Tucson, AZ: IEEE, pp.195–202. Available from: https://ieeexplore.ieee.org/document/4148934/ [accessed 22 November 2021].

Craig, A., O'Meley, P. and Carter, P. (2019). The Need for Greater Reporting of Medical Device Incidents. *EMJ Innovations*, 10 January 2019, pp.56–63.

Creswell, J.W. and Creswell, J.D. (2019). *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. 5th edn Thousand Oaks, California: SAGE Publications Ltd. Available from: https://www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=https://spada.uns.ac.id/pluginfile.php/510378/mod_resource/content/1/creswell.pdf&ved=2ahUKEwils6KChrCKAxU9UEEAHXkeEucQFnoECBsQAQ&usg=AOvVaw1BMQcHz-5FUg5agalH3wK7.

Crispin, L. (2006). Driving Software Quality: How Test-Driven Development Impacts Software Quality. *IEEE Software*, 23(6), pp.70–71.

Critical Software. (2020). *Role of Software in Medical Device Failures | Whitepaper* [online]. *Critical Software* [online]. Available from: https://criticalsoftware.com/en/resource/software-in-medical-device-failures [accessed 5 March 2024].

Crowe, S., Cresswell, K., Robertson, A., Huby, G., Avery, A. and Sheikh, A. (2011). The case study approach. *BMC Medical Research Methodology*, 11(1), p.100.

Czerwonka, J., Greiler, M. and Tilford, J. (2015). Code Reviews Do Not Find Bugs. How the Current Code Review Best Practice Slows Us Down. In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE). Florence, Italy: IEEE, pp.27–28. Available from: http://ieeexplore.ieee.org/document/7202946/ [accessed 18 August 2022].

Da Silva, C.E. and De Lemos, R. (2011). Dynamic plans for integration testing of self-adaptive software systems. In: *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. ICSE11: International Conference on Software Engineering. Waikiki, Honolulu HI USA: ACM, pp.148–157. Available from: https://dl.acm.org/doi/10.1145/1988008.1988029 [accessed 31 May 2023].

Daka, E. and Fraser, G. (2014). A Survey on Unit Testing Practices and Problems. In: *2014 IEEE 25th International Symposium on Software Reliability Engineering*. 2014 IEEE 25th International Symposium on Software Reliability Engineering (ISSRE). Naples, Italy: IEEE, pp.201–211. Available from: http://ieeexplore.ieee.org/document/6982627/ [accessed 12 July 2022].

Dalal, S. and Chhillar, R.S. (2013). Empirical study of root cause analysis of software failure. *SIGSOFT Softw. Eng. Notes*, 38(4), pp.1–7.

Dalalah, A. (2014). Extreme Programming: Strengths and Weaknesses. *Computer Technology and Application* [online], 5(1). Available from: http://www.davidpublisher.org/index.php/Home/Article/index?id=11751.html [accessed 5 January 2025].

D'Ambros, M., Bacchelli, A. and Lanza, M. (2010). On the Impact of Design Flaws on Software Defects. In: *2010 10th International Conference on Quality Software*. 2010 10th International Conference on Quality Software (QSIC). Zhangjiajie, China: IEEE, pp.23–31. Available from: http://ieeexplore.ieee.org/document/5562941/ [accessed 20 March 2023].

Damm, L.-O., Lundberg, L. and Olsson, D. (2005). Introducing Test Automation and Test-Driven Development: An Experience Report. *Electronic Notes in Theoretical Computer Science*, 116, pp.3–15.

Darwish, N.R. (2011). Improving the Quality of Applying eXtreme Programming (XP) Approach., 9(11).

Dasanayake, S., Aaramaa, S., Markkula, J. and Oivo, M. (2019). Impact of requirements volatility on software architecture: How do software teams keep up with ever-changing requirements? *Journal of Software: Evolution and Process*, 31(6), p.e2160.

Davison, R., Martinsons, M.G. and Kock, N. (2004). Principles of canonical action research. *Information Systems Journal*, 14(1), pp.65–86.

Deemer, P., Benefield, G., Larman, C. and Vodde, B. (2012). A Lightweight Guide to the Theory and Practice of Scrum., 2012.

DeJonckheere, M. and Vaughn, L.M. (2019). Semi-structured interviewing in primary care research: a balance of relationship and rigour. *Family Medicine and Community Health*, 7(2), p.e000057.

Delgado, D. and Martinez, A. (2013). Cost Effectiveness of Unit Testing: A Case Study in a Financial Institution. In: *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*. 2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement. pp.340–347.

Demissie, S., Keenan, F., Loughran, R. and McCaffery, F. (2020). Improving Multi-domain Stakeholder Communication of Embedded Safety-critical Development using Agile Practices: Expert Review. In: *Proceedings of the 8th International Conference on Model-Driven Engineering and Software Development*. Valletta, Malta: SCITEPRESS - Science and Technology Publications, pp.49– 56. Available from: http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0008977900490056 [accessed 11 January 2022].

Demissie, S., Keenan, F. and McCaffery, F. (2017). Supporting Embedded Medical Software Development with MDevSPICE® and Agile Practices., 11(2), p.5.

Despa, M.L. (2014). Comparative study on software development methodologies. *Database Systems Journal*, 5(3).

Di Penta, M., Bavota, G. and Zampetti, F. (2020). On the Relationship between Refactoring Actions and Bugs: A Differentiated Replication., 24 September 2020. Available from: http://arxiv.org/abs/2009.11685 [accessed 28 July 2022].

Diebold, P. (2013). *ACAPI - Agile Capability Analysis and Process Improvement in Highly Regulated Environments*. Kaiserslautern.

Diebold, P. and Dahlem, M. (2014). Agile practices in practice: a mapping study. In: *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*. EASE '14: 18th International Conference on Evaluation and Assessment in Software Engineering. London England United Kingdom: ACM, pp.1–10. Available from: https://dl.acm.org/doi/10.1145/2601248.2601254 [accessed 27 February 2023].

Digital.ai. (2023). *State of Agile* [online]. Available from: https://stateofagile.com/ [accessed 22 April 2025].

Digital.ai Agility. (2022). *16th Annual State of Agile Report*. Available from: https://info.digital.ai/rs/981-LQX-968/images/AR-SA-2022-16th-Annual-State-Of-Agile-Report.pdf.

Dingsøyr, T., Mikalsen, M., Solem, A. and Vestues, K. (2018). Learning in the Large - An Exploratory Study of Retrospectives in Large-Scale Agile Development. In: Garbajosa, J., Wang, X., and Aguiar, A., eds *Agile Processes in Software Engineering and Extreme Programming*. Lecture Notes in Business Information Processing. Cham: Springer International Publishing, pp.191–198. Available from: http://link.springer.com/10.1007/978-3-319-91602-6_13 [accessed 13 April 2023].

Dingsøyr, T. (2005). Postmortem reviews: purpose and approaches in software engineering. *Information and Software Technology*, 47(5), pp.293–303.

Dogša, T. and Batič, D. (2011). The effectiveness of test-driven development: an industrial case study. *Software Quality Journal*, 19(4), pp.643–661.

Doherty, B., Jelfs, A., Dasgupta, A. and Holden, P. (2016). Defect Analysis in Large Scale Agile Development: Quality in the Agile Factory Model. In: *2016 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA)*. 2016 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA). Berlin, Germany: IEEE, pp.180–180. Available from: http://ieeexplore.ieee.org/document/7809606/ [accessed 23 March 2023].

Donkoh, S. (2023). Application of triangulation in qualitative research. *Journal of Applied Biotechnology & Bioengineering*, 10(1), pp.6–9.

Doody, J. (2004). Strengthening the Practices of an Agile Methodology? 2004. Available from: https://arrow.dit.ie/itbj/vol5/iss1/16/ [accessed 22 March 2023].

Dorin, M. (2018). Coding Standards and Human Nature. *International Journal of Performability Engineering* [online], 2018. Available from: http://www.ijpe-online.com/EN/10.23940/ijpe.18.06.p22.13081313 [accessed 22 March 2023].

Dube, B., Nkomo, D. and Thokweng, M.A. (2024). Pragmatism: An Essential Philosophy for Mixed Methods Research in Education. *International Journal of Research and Innovation in Social Science*, VIII(III), pp.1001–1010.

Dudila, R. and Letia, I.A. (2013). Towards combining functional requirements tests and unit tests as a preventive practice against software defects. In: *2013 IEEE 9th International Conference on Intelligent Computer Communication and Processing (ICCP)*. 2013 IEEE International Conference on Intelligent Computer Communication and Processing (ICCP). Cluj-Napoca, Romania: IEEE, pp.279–282. Available from: http://ieeexplore.ieee.org/document/6646121/ [accessed 21 July 2022].

Dudovskiy, J. (2022). Action Research. *Research-Methodology* [online], 2022. Available from: https://research-methodology.net/research-methods/action-research/ [accessed 1 November 2022].

Dudziak, T. (2000). eXtreme Programming- An Overview., 2000. Available from: http//csis.pace.edu/~marchese/CS616/Agile/XP/XP_Overview.pdf [accessed 5 January 2025].

Duka, D. (2012). Agile Experiences in Software Development. In: *MIPRO, 2012 Proceedings of the 35th International Convention*. MIPRO, 2012 Proceedings of the 35th International Convention. p.6.

Durmus, M.S., Ustoglu, I., Tsarev, R.Y. and Börcsök, J. (2018). Enhanced V-Model. *Informatica* [online], 42(4). Available from: https://www.informatica.si/index.php/informatica/article/view/2027 [accessed 25 July 2023].

Eastwood, J., Ettema, R., De Souza, D., Liu, H., Busetto, L., Ben, H.-R., Patrick, H. and Schrijvers, G. (2018). Realist Research Design and Evaluation for Integrated Care RIC SIG - Part 1: Establishing a Special Interest Group. *International Journal of Integrated Care*, 18(s2), p.255.

Ebert, F., Castor, F., Novielli, N. and Serebrenik, A. (2021). An exploratory study on confusion in code reviews. *Empirical Software Engineering*, 26(1), p.12.

Edwards, S.H. (2003). Using Test-Driven Development in the Classroom: Providing Students with Automatic, Concrete Feedback on Performance. *Proc. Int'l Conf. Education and Information Systems: Technologies and Applications (EISTA 03)*, 2003, p.6.

Ellims, M., Bridges, J. and Ince, D.C. (2006). The Economics of Unit Testing. *Empirical Software Engineering*, 11(1), pp.5–31.

Ellims, M., Bridges, J. and Ince, D.C. (2004). Unit Testing in Practice. In: *15th International Symposium on Software Reliability Engineering*. 15th International Symposium on Software Reliability Engineering. Saint-Malo, Bretagne, France: IEEE, pp.3–13. Available from: http://ieeexplore.ieee.org/document/1383101/ [accessed 17 July 2022].

Erdoğan, O., Pekkaya, M.E. and Gök, H. (2018). More effective sprint retrospective with statistical analysis. *Journal of Software: Evolution and Process*, 30(5), p.e1933.

Erdogmus, H., Morisio, M. and Torchiano, M. (2005). On the effectiveness of the test-first approach to programming. *IEEE Transactions on Software Engineering*, 31(3), pp.226–237.

Ergasheva, S. and Kruglov, A. (2020). Software Development Life Cycle early phases and quality metrics: A Systematic Literature Review. *Journal of Physics: Conference Series*, 1694(1), p.012007.

EU. (2017). Regulation (EU) 2017/ 746 of the European Parliament and of the Council - of 5 April 2017 - on in vitro diagnostic medical devices and repealing Directive 98/79/ EC and Commission Decision 2010/227/EU., 2017.

EU. (2023). Regulation (EU) 2017/745 of the European Parliament and of the Council., 2023. Available from: https://eur-lex.europa.eu/eli/reg/2017/745/oj.

EU. (2022). The new SME definition: User guide and model declaration., 2022. Available from: https://www.eusmecentre.org.cn/wp-content/uploads/2022/12/SME-Definition.pdf.

European Parliament and the Council of the European Union. (1998). In Vitro Diagnostic Medical Device Directive (IVDMDD 98/79/EC)., 1998.

European Parliament, Council of the European Union. (2007). Council Directive 93/42/EEC., 2007.

Everhart, A.O., Karaca-Mandic, P., Redberg, R.F., Ross, J.S. and Dhruva, S.S. (2025). Late adverse event reporting from medical device manufacturers to the US Food and Drug Administration: cross sectional study. *BMJ*, 12 March 2025, p.e081518.

F. Tripp, J. and Armstrong, D.J. (2018). Agile Methodologies: Organizational Adoption Motives, Tailoring, and Performance. *Journal of Computer Information Systems*, 58(2), pp.170–179.

Fan, X. (2018). Orchestrating Agile Sprint Reviews in Undergraduate Capstone Projects. In: *2018 IEEE Frontiers in Education Conference (FIE)*. 2018 IEEE Frontiers in Education Conference (FIE). San Jose, CA, USA: IEEE, pp.1 – 8. Available from: https://ieeexplore.ieee.org/document/8658435/ [accessed 26 March 2023].

Faragó, C., Hegedűs, P. and Ferenc, R. (2015). Code Ownership: Impact on Maintainability. In: Gervasi, O. et al., eds *Computational Science and Its Applications -- ICCSA 2015*. Lecture Notes in Computer Science. Cham: Springer International Publishing, pp.3–19. Available from: https://link.springer.com/10.1007/978-3-319-21413-9_1 [accessed 13 February 2023].

FDA. (2018). 21 CFR PART 820-Quality System Regulation., 2018.

FDA. (2022a). Baxter recalls SIGMA Spectrum Infusion Pumps with Master Drug Library (Version 8) and Spectrum IQ Infusion Systems with Dose IQ Safety Software (Version 9) due to the risk of not alarming for repeated upstream occlusion events. *FDA* [online], 3 November 2022. Available from: https://www.fda.gov/medical-devices/medical-device-recalls/baxter-recalls-sigma-spectrum-infusion-pumps-master-drug-library-version-8-and-spectrum-iq-infusion [accessed 11 September 2023].

FDA. (2023a). *Medical Device Recalls form January 2023 to July 2023* [online]. Available from: https://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfRes/res.cfm?start_search=1&event_id=&productdescriptiontxt=&productcode=&IVDProducts=&rootCauseText=&recallstatus=&centerclassificationtypetext=&recallnumber=&postdatefrom=01%2F01%2F2023&postdateto=07%2F25%2F2023&productshortreasontxt=software&firmlegalnam=&PMA_510K_Num=&pnumber=&knumber=&sortcolumn=cca [accessed 26 July 2023].

FDA. (2025a). Outpatient Telemetry Correction: Philips Issues Correction for Monitoring Service Application used with Mobile Cardiac Outpatient Telemetry Due to Potential for Missed Information or Notifications That May Impact Patient Care. *FDA* [online], 14 January 2025. Available from: https://www.fda.gov/medical-devices/medical-device-recalls/outpatient-telemetry-correction-philips-issues-correction-monitoring-service-application-used-mobile [accessed 20 March 2025].

FDA. (2023b). *Premarket Approval (PMA)* [online]. *US FDA* [online]. Available from: https://www.fda.gov/medical-devices/premarket-submissions-selecting-and-preparing-correct-submission/premarket-approval-pma [accessed 23 September 2025].

FDA. (2002). Premarket Approval (PMA)., October 2002.

FDA. (2024a). *Premarket Notification 510(k)* [online]. *FDA* [online]. Available from: https://www.fda.gov/medical-devices/premarket-submissions-selecting-and-preparing-correct-submission/premarket-notification-510k [accessed 23 September 2025].

FDA. (2015). Premarket Notification 510(k)., 2015.

FDA. (2024b). Quality Management System Regulation: Final Rule Amending the Quality System Regulation – Frequently Asked Questions. *FDA* [online], 7 August 2024. Available from: https://www.fda.gov/medical-devices/quality-system-qs-regulationmedical-device-

current-good-manufacturing-practices-cgmp/quality-management-system-regulation-final-rule-amending-quality-system-regulation-frequently-asked [accessed 23 September 2025].

FDA. (2025b). *Recalls, Corrections and Removals (Devices) | FDA* [online]. Available from: https://www.fda.gov/medical-devices/postmarket-requirements-devices/recalls-corrections-and-removals-devices [accessed 30 July 2025].

FDA. (2022b). Vyaire Medical Recalls bellavista 1000 and 1000e Series Ventilators Due to Issues with Software Configurations. *FDA* [online], 17 February 2022. Available from: https://www.fda.gov/medical-devices/medical-device-recalls/vyaire-medical-recalls-bellavista-1000-and-1000e-series-ventilators-due-issues-software [accessed 11 September 2023].

FDA, C. for D. and R. (2020). Digital Health Criteria. *FDA* [online], 22 September 2020. Available from: https://www.fda.gov/medical-devices/digital-health-center-excellence/digital-health-criteria [accessed 17 October 2021].

Felidré, W., Furtado, L., da Costa, D., Cartaxo, B. and Pinto, G. (2019). Continuous Integration Theater., 2 July 2019. Available from: http://arxiv.org/abs/1907.01602 [accessed 21 September 2022].

Felix, E.A. and Lee, S.P. (2020). Predicting the number of defects in a new software version. *PLoS ONE*, 15(3), p.e0229131.

Ferreira, I., Fernandes, E., Cedrim, D., Uchôa, A., Bibiano, A.C., Garcia, A., Correia, J.L., Santos, F., Nunes, G., Barbosa, C., Fonseca, B. and de Mello, R. (2018). The buggy side of code refactoring: understanding the relationship between refactorings and bugs. In: *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. ICSE '18: 40th International Conference on Software Engineering. Gothenburg Sweden: ACM, pp.406– 407. Available from: https://dl.acm.org/doi/10.1145/3183440.3195030 [accessed 27 July 2022].

Fitzgerald, B., Stol, K.-J., O'Sullivan, R. and O'Brien, D. (2013). Scaling agile methods to regulated environments: An industry case study. In: *2013 35th International Conference on Software Engineering (ICSE)*. 2013 35th International Conference on Software Engineering (ICSE). San Francisco, CA, USA: IEEE, pp.863 – 872. Available from: http://ieeexplore.ieee.org/document/6606635/ [accessed 28 October 2021].

Flohr, T. and Schneider, T. (2006). Lessons Learned from an XP Experiment with Students: Test-First Needs More Teachings. In: Münch, J. and Vierimaa, M., eds *Product-Focused Software Process Improvement*. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, pp.305–318.

Fogarty, A., Edgeworth, A., Smith, O., Dowling, M., Yilmaz, M., MacMahon, S.T. and Clarke, P. (2020). Agile Software Development – Do We Really Calculate the Costs? A Multivocal Literature Review. In: Yilmaz, M. et al., eds *Systems, Software and Services Process Improvement*. Communications in Computer and Information Science. Cham: Springer International Publishing, pp.203–219. Available from: https://link.springer.com/10.1007/978-3-030-56441-4_15 [accessed 14 May 2023].

Foucault, M., Falleri, J.-R. and Blanc, X. (2014). Code ownership in open-source software. In: *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*. EASE '14: 18th International Conference on Evaluation and Assessment in Software Engineering. London England United Kingdom: ACM, pp.1–9. Available from: https://dl.acm.org/doi/10.1145/2601248.2601283 [accessed 13 February 2023].

Fowler, M. (2006). *Code Ownership* [online]. *martinfowler.com* [online]. Available from: https://martinfowler.com/bliki/CodeOwnership.html [accessed 13 February 2023].

Freimut, B. (2001). *Developing and Using Defect Classification Schemes* [unpublished].

French-Mowat, E. and Burnett, J. (2012). How are medical devices regulated in the European Union? *Journal of the Royal Society of Medicine*, 105(Suppl 1), pp.S22–S28.

Fruhling, A., Tyser, K. and De Vreede, G.-J. (2005). Experiences with Extreme Programming in Telehealth: Developing and Implementing a Biosecurity Health Care Application. In: *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*. 38th Annual Hawaii International Conference on System Sciences. Big Island, HI, USA: IEEE, pp.151b–151b. Available from: http://ieeexplore.ieee.org/document/1385543/ [accessed 2 January 2025].

Fu, K. (2011). Trustworthy Medical Device Software., 11 April 2011, p.20.

Fu, Q., Grady, F., Broberg, B.F., Roberts, A., Martens, G.G., Johansen, K.V. and Le Loher, P. (2017). Code Review and Cooperative Pair Programming Best Practice. *International Journal of Software Engineering & Applications*, 8(4), pp.13–19.

Fu, Z., Guo, C., Zhang, Z., Ren, S., Jiang, Y. and Sha, L. (2017). Study of Software-Related Causes in the FDA Medical Device Recalls. In: *2017 22nd International Conference on Engineering of Complex Computer Systems (ICECCS)*. 2017 22nd International Conference on Engineering of Complex Computer Systems (ICECCS). Fukuoka: IEEE, pp.60–69. Available from: https://ieeexplore.ieee.org/document/8292803/ [accessed 22 February 2022].

Gabriel, D. (2013). *Inductive and deductive approaches to research | Dr Deborah Gabriel* [online]. Available from: https://deborahgabriel.com/2013/03/17/inductive-and-deductive-approaches-to-research/ [accessed 16 November 2022].

Gary, K., Enquobahrie, A., Ibanez, L., Cheng, P., Yaniv, Z., Cleary, K., Kokoori, S., Muffih, B. and Heidenreich, J. (2011). Agile methods for open source safety-critical software. *Software: Practice and Experience*, 41(9), pp.945–962.

Gatrell, M. and Counsell, S. (2015). The effect of refactoring on change and fault-proneness in commercial C# software. *Science of Computer Programming*, 102, pp.44–56.

Geambaşu, C.V., Jianu, Iulia, Jianu, Ionel and Gavrilă, A. (2011). Influence Factors for the Choice of a Software Development Methodology., 10(4).

George, B. and Williams, L. (2004). A structured experiment of test-driven development. *Information and Software Technology*, 46(5), pp.337–342.

Geppert, B., Mockus, A. and Rossler, F. (2005). Refactoring for Changeability: A Way to Go? In: *11th IEEE International Software Metrics Symposium (METRICS'05)*. 11th IEEE

International Software Metrics Symposium (METRICS'05). Como, Italy: IEEE, pp.13–13. Available from: http://ieeexplore.ieee.org/document/1509291/ [accessed 4 August 2022].

Geraldi, R.T. and OliveiraJr, E. (2017). Defect Types and Software Inspection Techniques: A Systematic Mapping Study. *Journal of Computer Science*, 13(10), pp.470–495.

Glaiel, F., Moulton, A. and Madnick, S. (2014). Agile Project Dynamics: A System Dynamics Investigation of Agile Software Development Methods., October 2014, p.31.

Glasow, P. (2005). Fundamentals of Survey Research Methodology., April 2005, p.28.

Gokarna, M. and Singh, R. (2021). DevOps: A Historical Review and Future Works. In: *2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*. 2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS). Greater Noida, India: IEEE, pp.366 – 371. Available from: https://ieeexplore.ieee.org/document/9397235/ [accessed 12 February 2025].

Golafshani, N. (2015). Understanding Reliability and Validity in Qualitative Research. *The Qualitative Report* [online], 23 January 2015. Available from: https://nsuworks.nova.edu/tqr/vol8/iss4/6/ [accessed 11 August 2025].

Golfarelli, M., Rizzi, S. and Turricchia, E. (2012). Sprint Planning Optimization in Agile Data Warehouse Design. In: Cuzzocrea, A. and Dayal, U., eds *Data Warehousing and Knowledge Discovery*. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, pp.30–41. Available from: http://link.springer.com/10.1007/978-3-642-32584-7_3 [accessed 26 March 2023].

Gordon, W. and Stern, A.D. (2019). Challenges and opportunities in software-driven medical devices. *Nature Biomedical Engineering*, 3. Available from: http://www.nature.com/articles/s41551-019-0426-z.

Görg, C. and Weißgerber, P. (2005). Error detection by refactoring reconstruction. In: *Proceedings of the 2005 international workshop on Mining software repositories*. MSR '05. New York, NY, USA: Association for Computing Machinery, pp.1–5. Available from: https://doi.org/10.1145/1083142.1083148 [accessed 29 July 2022].

Goundar, S. (2012). Research Methodology-A2. *Research Methodology*, May 2012, p.47.

Granlund, T., Stirbu, V. and Mikkonen, T. (2021). Towards Regulatory-Compliant MLOps: Oravizio's Journey from a Machine Learning Experiment to a Deployed Certified Medical Product. *SN Computer Science*, 2(5), p.342.

Grant, M.J. and Booth, A. (2009). A typology of reviews: an analysis of 14 review types and associated methodologies. *Health Information & Libraries Journal*, 26(2), pp.91–108.

Greiler, M., Herzig, K. and Czerwonka, J. (2015). Code Ownership and Software Quality: A Replication Study. In: *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories (MSR). Florence, Italy: IEEE, pp.2– 12. Available from: http://ieeexplore.ieee.org/document/7180062/ [accessed 13 February 2023].

Gren, L. and Antinyan, V. (2017). On the Relation Between Unit Testing and Code Quality. In: *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA). pp.52–56.

Grisham, P.S. and Perry, D.E. (2005). Customer Relationships and Extreme Programming., 30(4), pp.1–6.

Guo, Z. and Hirschmann, C. (2012). An Integrated Process for Developing Safety-critical Systems using Agile Development Methods., 2012, p.3.

Gupta, N. and Sahu, M. (2024). Problem Faced During the Software Development Cycle. *International Journal of Scientific Research in Engineering and Management*, 08(06), pp.1–5.

Gupta, N.K. and Rohil, M.K. (2015). An Approach for Detection and Correction of Design Defects in Object Oriented Software., 2015, p.6.

Haj-Bolouri, A., Purao, S., Rossi, M. and Bernhardsson, L. (2017). Action Design Research as a Method-in-Use: Problems and Opportunities., 2017, p.9.

Haleem, Mohd., Farooqui, Md.F. and Faisal, Md. (2021). Tackling Requirements Uncertainty in Software Projects: A Cognitive Approach. *International Journal of Cognitive Computing in Engineering*, 2, pp.180–190.

Han, X., Tahir, A., Liang, P., Counsell, S., Blincoe, K., Li, B. and Luo, Y. (2022). Code Smells Detection via Modern Code Review: A Study of the OpenStack and Qt Communities., 16 May 2022. Available from: http://arxiv.org/abs/2205.07535 [accessed 18 August 2022].

Harlacher, J. (2016). *An educator's guide to questionnaire development*. Washington, DC: U.S. Department of Education, Institute of Education Sciences, National Center for Education Evaluation and Regional Assistance, Regional Educational Laboratory Central. Available from: https://eric.ed.gov/?id=ED562653.

Hassmén, P., Keegan, R. and Piggott, D. (2016). Research Paradigms, Methodologies and Methods. In: *Rethinking Sport and Exercise Psychology Research*. London: Palgrave Macmillan UK, pp.105–129. Available from: http://link.springer.com/10.1057/978-1-137-48338-6_5 [accessed 18 December 2024].

Hauschild, A.-C., Martin, R., Holst, S.C., Wienbeck, J. and Heider, D. (2022). Guideline for software life cycle in health informatics. *iScience*, 25(12), p.105534.

Haynes, S.R. and Friedenberg, M. (2006). Best Practices in Agile Software Development., April 2006.

Heeager, L.T. (2014). How Can Agile and Documentation-Driven Methods be Meshed in Practice? In: Cantone, G. and Marchesi, M., eds *Agile Processes in Software Engineering and Extreme Programming*. Lecture Notes in Business Information Processing. Cham: Springer International Publishing, pp.62–77. Available from: http://link.springer.com/10.1007/978-3-319-06862-6_5 [accessed 5 December 2021].

Heeager, L.T. and Nielsen, P.A. (2009). Agile Software Development and its Compatibility with a Document-Driven Approach? A Case Study., 2009, p.11.

Heeager, L.T. and Nielsen, P.A. (2020). Meshing agile and plan-driven development in safety-critical software: a case study. *Empirical Software Engineering*, 25(2), pp.1035–1062.

Hema, V., Thota, S., Naresh Kumar, S., Padmaja, C., Rama Krishna, C.B. and Mahender, K. (2020). Scrum: An Effective Software Development Agile Tool. *IOP Conference Series: Materials Science and Engineering*, 981(2), p.022060.

Hennink, M. and Kaiser, B.N. (2022). Sample sizes for saturation in qualitative research: A systematic review of empirical tests. *Social Science & Medicine*, 292, p.114523.

Hevner, A.R., March, S.T., Park, J. and Ram, S. (2004). Design Science in Information Systems Research., 2004, p.32.

Highsmith, J. (2000). *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems* [online]. Dorset House Publishing. Available from: https://docs.adaptdev.info/lib/XKDXINZ7 [accessed 15 January 2025].

Highsmith, J. (2002). *Agile software development ecosystems*. USA: Addison-Wesley Longman Publishing Co., Inc. Available from: https://dl.acm.org/doi/book/10.5555/513727.

Hiller, J. (2016). Epistemological Foundations of Objectivist and Interpretivist Research., 2016, p.29.

Hilton, M., Nelson, N., Tunnell, T., Marinov, D. and Dig, D. (2017). Trade-offs in continuous integration: assurance, security, and flexibility. In: *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ESEC/FSE 2017. New York, NY, USA: Association for Computing Machinery, pp.197– 207. Available from: https://doi.org/10.1145/3106237.3106270 [accessed 23 September 2022].

Hilton, M., Tunnell, T., Huang, K., Marinov, D. and Dig, D. (2016). Usage, costs, and benefits of continuous integration in open-source projects. In: *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. ASE'16: ACM/IEEE International Conference on Automated Software Engineering. Singapore: ACM, pp.426– 437. Available from: https://dl.acm.org/doi/10.1145/2970276.2970358 [accessed 25 September 2022].

Hoda, R., Noble, J. and Marshall, S. (2010). Agile Undercover: When Customers Don't Collaborate. In: Sillitti, A. et al., eds *Agile Processes in Software Engineering and Extreme Programming*. Lecture Notes in Business Information Processing. Berlin, Heidelberg: Springer, pp.73–87.

Holland, A. (2024). The Quality Management System Regulation (QMSR): Implications for Manufacturers of Medical Devices., October 2024.

Holling, D., Hofbauer, A., Pretschner, A. and Gemmar, M. (2016). Profiting from Unit Tests for Integration Testing. In: *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. 2016 IEEE International Conference on Software Testing, Verification and Validation (ICST). Chicago, IL, USA: IEEE, pp.353–363. Available from: http://ieeexplore.ieee.org/document/7515486/ [accessed 31 January 2023].

Hong, J. and Cross Francis, D. (2020). Unpacking complex phenomena through qualitative inquiry: The case of teacher identity research. *Educational Psychologist*, 55(4), pp.208–219.

Hovorushchenko, T. and Popov, P. (2021). Method of Developing the Defect-Free Medical Software by Establishing the Presence of Residual Defects. In: *4th International Conference on Informatics & Data-Driven Medicine*. Informatics & Data-Driven Medicine.

Hrgarek, N. (2012). Certification and regulatory challenges in medical device software development. In: *2012 4th International Workshop on Software Engineering in Health Care (SEHC)*. pp.40–43.

Huang, F. and Liu, B. (2017). Software defect prevention based on human error theories. *Chinese Journal of Aeronautics*, 30(3), pp.1054–1070.

Hugo Santiago C. Pinto, V., R. Oliveira, R., F. Vilela, R. and R. S. Souza, S. (2018). Evaluating the User Acceptance Testing for Multi-tenant Cloud Applications. In: *Proceedings of the 8th International Conference on Cloud Computing and Services Science*. 8th International Conference on Cloud Computing and Services Science. Funchal, Madeira, Portugal: SCITEP RESS - Science and Technology Publications, pp.47 – 56. Available from: http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0006664000470056 [accessed 3 April 2023].

Hulkko, H. and Abrahamsson, P. (2005). A multiple case study on the impact of pair programming on product quality. In: *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.* Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005. pp.495–504.

Hunt, J., ed. (2006). *Agile Software Construction*. London: Springer London.

Hyman, M.R. and Sierra, J.J. (2016). Open- versus Close-Ended Survey Questions. *Business Outlook*, 14(2).

Ibec. (2016). Irish Medtech 2020 Strategy the Global Hub., February 2016. Available from: https://www.ibec.ie/-/media/documents/connect-and-learn/industries/life-sciences-and-healthcare/irish-medtech-association/irish-medtech-2020-strategy-the-global-hub.pdf.

Ibec. (2022). *Medtech Strategy 2025 - IBEC* [online]. Available from: https://www.ibec.ie/connect-and-learn/industries/life-sciences-and-healthcare/medtech-strategy-2025 [accessed 27 November 2023].

Ibrahim, N. (2007). An Overview of Agile Software Development Methodology and Its Relevance to Software Engineering., 2(1), p.12.

Igwenagu, C. (2016). *Fundamentals of research methodology and data collection*.

Iivari, J. and Venable, J.R. (2009). Action research and design science research - Seemingly similar but decisively dissimilar., 2009, p.13.

IMDRF. (2014). 'Software as a Medical Device': Possible Framework for Risk Categorization and Corresponding Considerations., 2014, p.30.

Inayat, I., Noor, M.A. and Inayat, Z. (2012). Facilitating an Off-Site Customer in Product-Based Agile Software Development: An Industrial Case Study. In: Chowdhry, B. S. et al., eds *Emerging Trends and Applications in Information Communication Technologies*. Communications in Computer and Information Science. Berlin, Heidelberg: Springer Berlin

Heidelberg, pp.210–221. Available from: http://link.springer.com/10.1007/978-3-642-28962-0_21 [accessed 6 January 2025].

Irshad, M., Britto, R. and Petersen, K. (2021). Adapting Behavior Driven Development (BDD) for large-scale software systems. *Journal of Systems and Software*, 177, p.110944.

ISO. (2019). *ISO 14971:2019(en), Medical devices — Application of risk management to medical devices* [online]. Available from: https://www.iso.org/obp/ui/#iso:std:iso:14971:ed-3:v1:en [accessed 31 July 2023].

ISO. (2015). ISO 62366-1-2015 -Application of usability engineering to medical devices.

ISO. (2016). *ISO 13485:2016(en), Medical devices — Quality management systems — Requirements for regulatory purposes* [online]. Available from: https://www.iso.org/obp/ui/en/#iso:std:iso:13485:ed-3:v1:en [accessed 31 July 2023].

Ivanova, D. and Kadurin, V. (2021). A new proposed software development methodology for healthcare industry. In: Tomsk, Russia, p.030010. Available from: http://aip.scitation.org/doi/abs/10.1063/5.0042261 [accessed 30 October 2021].

Jain, R. and Suman, U. (2016). Effectiveness of Agile Practices in Global Software Development. *International Journal of Grid and Distributed Computing*, 9(10), pp.231–248.

Jalote, P. and Haragopal, M. (1998). Overcoming the NAH syndrome for inspection deploym ent. In: *Proceedings of the 20th International Conference on Software Engineering*. 20th International Conference on Software Engineering. Kyoto, Japan: IEEE Comput. Soc, pp.371–378. Available from: http://ieeexplore.ieee.org/document/671390/ [accessed 19 July 2022].

Jamal, N., Zulqarnain, M., Boota, M.W., Khan, S. and Akber, S.M.A. (2015). Role of Defect Prevention Techniques Vs Defect Detection to Improve Software Quality: Critical Analysis Summary of Defect Preventive Approaches., 2(3).

Jamshed, S. (2014). Qualitative research method-interviewing and observation. *Journal of Basic and Clinical Pharmacy*, 5(4), p.87.

Jansen, H. (2010). The Logic of Qualitative Survey Research and its Position in the Field of Social Research Methods., 11(2). Available from: https://www.qualitative-research.net/index.php/fqs/article/view/1450.

Jansen, T., Khalil, Z.A., Constantinou, E. and Mens, T. (2021). Does the duration of rapid release cycles affect the bug handling activity? 15 March 2021. Available from: http://arxiv.org/abs/2103.08648 [accessed 15 February 2023].

Janzen, D.S. (2005). Software architecture improvement through test-driven development. In: *Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. OOPSLA ’05. New York, NY, USA: Association for Computing Machinery, pp.222– 223. Available from: https://doi.org/10.1145/1094855.1094945 [accessed 27 June 2022].

Janzen, D.S. and Saiedian, H. (2007). A Leveled Examination of Test-Driven Development Acceptance. In: *29th International Conference on Software Engineering (ICSE’07)*. 29th International Conference on Software Engineering (ICSE’07). pp.719–722.

Janzen, D.S. and Saiedian, H. (2006). On the Influence of Test-Driven Development on Software Design. In: *19th Conference on Software Engineering Education & Training (CSEE T'06)*. 19th Conference on Software Engineering Education & Training (CSEET'06). pp.141–148.

Jeeva Padmini, K.V., Perera, I. and Dilum Bandara, H.M.N. (2016). Applying agile practices to avoid chaos in User Acceptance Testing: A case study. In: *2016 Moratuwa Engineering Research Conference (MERCon)*. 2016 Moratuwa Engineering Research Conference (MERCon). Moratuwa, Sri Lanka: IEEE, pp.96–101. Available from: http://ieeexplore.ieee.org/document/7480122/ [accessed 30 March 2023].

Jensen, R. (2003). A pair programming experience. *undefined* [online], 2003. Available from: https://www.semanticscholar.org/paper/A-pair-programming-experience-Jensen/5d9e87bd074af77ea19be939514a90ad2c4bd3bd [accessed 15 May 2022].

Jetley, R. and Chelf, B. (2009). Diagnosing medical device software defects using static analy sis., 31.

Jetley, R.P., Jones, P.L. and Anderson, P. (2008). Static analysis of medical device software using CodeSonar. In: *Proceedings of the 2008 workshop on Static analysis*. SAW '08. New York, NY, USA: Association for Computing Machinery, pp.22–29. Available from: https://doi.org/10.1145/1394504.1394507 [accessed 11 September 2023].

Jiang, S., Zhang, M., Zhang, Y., Wang, R., Yu, Q. and Keung, J. (2021). An Integration Test Order Strategy to Consider Control Coupling. *IEEE Transactions on Software Engineering*, 47(7), pp.1350–1367.

Johannesson, P. and Perjons, E. (2014). Research Strategies and Methods. In: *An Introduction to Design Science*. Cham: Springer International Publishing, pp.39–73. Available from: http://link.springer.com/10.1007/978-3-319-10632-8_3 [accessed 21 November 2022].

Johnson, B., Song, Y., Murphy-Hill, E. and Bowdidge, R. (2013). Why don't software developers use static analysis tools to find bugs? In: *2013 35th International Conference on Software Engineering (ICSE)*. 2013 35th International Conference on Software Engineering (ICSE). pp.672–681.

Johnsson, V., Tolsgaard, M.G., Petersen, O.B.B. and Svendsen, M.B.S. (2023). Building low-cost simulators for invasive ultrasound-guided procedures using the V-model. *Advances in Simulation*, 8(1), pp.1–9.

Jonsson, H., Larsson, S. and Punnekkat, S. (2012). Agile Practices in Regulated Railway Software Development. In: *2012 IEEE 23rd International Symposium on Software Reliability Engineering Workshops*. 2012 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). Dallas, TX, USA: IEEE, pp.355–360. Available from: http://ieeexplore.ieee.org/document/6405469/ [accessed 21 March 2023].

Kabbedijk, J., Brinkkemper, S., Jansen, S. and van der Veldt, B. (2009). Customer Involvement in Requirements Management: Lessons from Mass Market Software Development. In: *2009 17th IEEE International Requirements Engineering Conference*. 2009 17th IEEE International Requirements Engineering Conference (RE). Atlanta, Georgia, USA: IEEE, pp.281–286. Available from: http://ieeexplore.ieee.org/document/5328527/ [accessed 28 September 2022].

Kahles, A., Goldschmid, H., Volckmar, A.-L., Ploeger, C., Kazdal, D., Penzel, R., Budczies, J., Flechtenmacher, C., Gassner, U.M., Brüggemann, M., Vogeser, M., Schirmacher, P. and Stenzinger, A. (2023). Regulation (EU) 2017/746 (IVDR): practical implementation of annex I in pathology. *Die Pathologie*, 44(S2), pp.86–95.

Kallio, H., Pietilä, A., Johnson, M. and Kangasniemi, M. (2016). Systematic methodological review: developing a framework for a qualitative semi-structured interview guide. *Journal of Advanced Nursing*, 72(12), pp.2954–2965.

Kamalrudin, M., Ow, L.L. and Sidek, S. (2018). Requirements Defects Techniques in Requirements Analysis: A Review., 10(1). Available from: https://jtec.utem.edu.my/jtec/article/download/3594/2694.

Kane, D.W., Hohman, M.M., Cerami, E.G., McCormick, M.W., Kuhlmman, K.F. and Byrd, J.A. (2006). Agile methods in biomedical software development: a multi-site experience report. *BMC Bioinformatics*, 7(1), p.273.

Kannan, V., Basit, M.A., Bajaj, P., Carrington, A.R., Donahue, I.B., Flahaven, E.L., Medford, R., Melaku, T., Moran, B.A., Saldana, L.E., Willett, D.L., Youngblood, J.E. and Toomay, S.M. (2019). User stories as lightweight requirements for agile clinical decision support development. *Journal of the American Medical Informatics Association*, 26(11), pp.1344–1354.

Kannangara, S.H. (2013). Impact of Refactoring on Code Quality Improvement in Software Maintenance., 2, p.6.

Karlesky, M.J., Bereza, W.I. and Erickson, C.B. (2006). Effective Test Driven Development for Embedded Software. In: *2006 IEEE International Conference on Electro/Information Technology*. 2006 IEEE International Conference on Electro/Information Technology. pp.382–387.

Kaushik, V. and Walsh, C.A. (2019). Pragmatism as a Research Paradigm and Its Implications for Social Work Research. *Social Sciences*, 8(9), p.255.

Keskin Kaynak, İ., Çilden, E. and Aydin, S. (2019). Software Quality Improvement Practices in Continuous Integration. In: Walker, A., O'Connor, R. V., and Messnarz, R., eds *Systems, Software and Services Process Improvement*. Communications in Computer and Information Science. Cham: Springer International Publishing, pp.507–517.

Khan, A.I., Qureshi, M.R.J. and Khan, U.A. (2012). A Comprehensive Study of Commonly Practiced Heavy & Light Weight Software Methodologies., 2012. Available from: https://arxiv.org/abs/1202.2514 [accessed 23 December 2024].

Khan, S.M.A. (2023). Waterfall Model Used in Software Development Reference: Software Requirements Engineering Waterfall Model., 2023. Available from: https://rgdoi.net/10.13140/RG.2.2.29580.69764 [accessed 25 December 2024].

Khanam, Z. and Ahsan, M.N. (2017). Evaluating the effectiveness of test driven development: Advantages and pitfalls. *International Journal of Applied Engineering Research*, 12, pp.7705–7716.

Khomh, F., Dhaliwal, T., Zou, Y. and Adams, B. (2012). Do faster releases improve software quality? An empirical case study of Mozilla Firefox. In: *2012 9th IEEE Working Conference*

on Mining Software Repositories (MSR). 2012 9th IEEE Working Conference on Mining Software Repositories (MSR 2012). Zurich: IEEE, pp.179–188. Available from: https://ieeexplore.ieee.org/document/6224279/ [accessed 15 February 2023].

Kim, G., Debois, P., Willis, J., Humble, J. and Allspaw, J. (2016). *The DevOps handbook: how to create world-class agility, reliability, & security in technology organizations*. First edition. Portland, OR: IT Revolution Press, LLC.

Kim, M., Zimmermann, T. and Nagappan, N. (2012). A field study of refactoring challenges and benefits. In: *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering - FSE '12*. the ACM SIGSOFT 20th International Symposium. Cary, North Carolina: ACM Press, p.1. Available from: http://dl.acm.org/citation.cfm?doid=2393596.2393655 [accessed 26 July 2022].

Kircher, M. and Hofman, P. (2012). Combining systematic reuse with Agile development: experience report. In: *Proceedings of the 16th International Software Product Line Conference - Volume 1*. SPLC '12. New York, NY, USA: Association for Computing Machinery, pp.215–219. Available from: https://doi.org/10.1145/2362536.2362566 [accessed 8 January 2022].

Kitawaki, R., Umezu, M., Iwasaki, K. and Kasanuki, H. (2016). Analysis of Medical Device Recalls Owing to Output Information from Software. *undefined* [online], 2016. Available from: https://www.semanticscholar.org/paper/Analysis-of-Medical-Device-Recalls-Owing-to-Output-Kitawaki-Umezu/e913ee65b2147ac3ff5677b4d91443de8b25d8b3 [accessed 21 October 2021].

Kivunja, C. and Kuyini, A.B. (2017). Understanding and Applying Research Paradigms in Educational Contexts. *International Journal of Higher Education*, 6(5), p.26.

Klammer, C. and Kern, A. (2015). Writing unit tests: It's now or never! In: *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. 2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW). Graz, Austria: IEEE, pp.1–4. Available from: http://ieeexplore.ieee.org/document/7107469/ [accessed 18 July 2022].

Knight, J.C. and Wika, K.G. (2010). Software Safety in Medical Applications. *Computer Aided Surgery*, 1(3), pp.121–132.

Koi-Akrofi, G.Y., Akrofi, J.K. and Akwetey Matey, H. (2019). Understanding the Characteristics, Benefits and Challenges of Agile it Project Management: A Literature Based Perspective. *International Journal of Software Engineering & Applications*, 10(5), pp.25–44.

Kokkola, M. (2004). Arranging defect management in an agile development process., 2004, p.9.

Kononenko, O., Baysal, O., Guerrouj, L., Cao, Y. and Godfrey, M.W. (2015). Investigating code review quality: Do people and participation matter? In: *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME). Bremen, Germany: IEEE, pp.111–120. Available from: http://ieeexplore.ieee.org/document/7332457/ [accessed 25 August 2022].

Korkala, M., Abrahamsson, P. and Kyllonen, P. (2006). A Case Study on the Impact of Customer Communication on Defects in Agile Software Development. In: *AGILE 2006 (AGILE'06)*. AGILE 2006 (AGILE'06). Minneapolis, MN, USA: IEEE, pp.76–88. Available from: http://ieeexplore.ieee.org/document/1667565/ [accessed 21 March 2022].

Kortum, F., Klünder, J., Brunotte, W. and Schneider, K. (2019). Sprint Performance Forecasts in Agile Software Development - The Effect of Futurespectives on Team-Driven Dynamics. In: The 31st International Conference on Software Engineering and Knowledge Engineering. pp.94 – 101. Available from: http://ksiresearchorg.ipage.com/seke/seke19paper/seke19paper_224.pdf [accessed 26 March 2023].

Koskela, J., Myllyaho, M., Kääriäinen, J., Bendas, D., Hyysalo, J. and Virta, A. (2004). Experiences of Using Extreme Programming to Support a Legacy Information System Migration Project., 2004.

Kosti, M.D. (2017). Challenges of Agile Practices Implementation in the Medical Device Software Development Methodologies., 7(2), p.9.

Kotagi, V. and Yadav, S.K. (2023). Defect Analysis in Requirements Testing Phase of Software Development Life Cycle., 25 July 2023. Available from: https://papers.ssrn.com/abstract=4520332 [accessed 24 February 2024].

Kouroshfar, E., Mirakhorli, M., Bagheri, H., Xiao, L., Malek, S. and Cai, Y. (2015). A Study on the Role of Software Architecture in the Evolution and Quality of Software. In: *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories (MSR). Florence, Italy: IEEE, pp.246–257. Available from: http://ieeexplore.ieee.org/document/7180084/ [accessed 7 March 2024].

Kruchten, P. (2000). The Rational Unified Process--An Introduction. In: p.255.

Kumar, M. and Dwivedi, D.R.K. (2021). Agile Modeling with Extreme Programming: Values, Principles, and Practices., 08(01).

Laanti, M. and Kettunen, P. (2019). SAFe Adoptions in Finland: A Survey Research. In: Hoda, R., ed. *Agile Processes in Software Engineering and Extreme Programming – Workshops*. Lecture Notes in Business Information Processing. Cham: Springer International Publishing, pp.81–87. Available from: http://link.springer.com/10.1007/978-3-030-30126-2_10 [accessed 22 April 2025].

Lai, S.-T. (2015). A Maintainability Enhancement Procedure for Reducing Agile Software Development Risk. *International Journal of Software Engineering & Applications*, 6(4), pp.29–40.

Larman, C. and Vodde, B. (2017). *Large-scale Scrum: more with LeSS*. Boston: Addison-Wesley.

Latorre, R. (2014). A successful application of a Test-Driven Development strategy in the industrial environment. *Empirical Software Engineering*, 19(3), pp.753–773.

Lau, F. (1999). Toward a framework for action research in information systems studies. *Information Technology & People*, 12(2), pp.148–176.

Lau, F. and Kuziemsky, C. (2016). *Handbook of eHealth Evaluation: An Evidence-based Approach*. Victoria, British Columbia Canada V8P 5C2: University of Victoria. Available from: https://www.ncbi.nlm.nih.gov/books/NBK481583/.

Layman, L., Williams, L. and Cunningham, L. (2004). Exploring Extreme Programming in Context: An Industrial Case Study. In: *Agile Development Conference*. Agile Development Conference. Salt Lake City, UT, USA: IEEE, pp.32–41. Available from: http://ieeexplore.ieee.org/document/1359793/ [accessed 11 May 2022].

Lee, I., Pappas, G., Cleaveland, R., Hatcliff, J., Krogh, B., Lee, P., Rubin, H. and Sha, L. (2006). High-Confidence Medical Device Software and Systems. *Computer*, 39, pp.33–38.

Lehtinen, T.O.A., Virtanen, R., Viljanen, J.O., Mäntylä, M.V. and Lassenius, C. (2014). A tool supporting root cause analysis for synchronous retrospectives in distributed software teams. *Information and Software Technology*, 56(4), pp.408–437.

Lehtinen, T.O.A., Mäntylä, M.V., Itkonen, J. and Vanhanen, J. (2015). Diagrams or structural lists in software project retrospectives – An experimental comparison. *Journal of Systems and Software*, 103, pp.17–35.

Lehtinen, T.O.A., Itkonen, J. and Lassenius, C. (2017). Recurring opinions or productive improvements—what agile teams actually discuss in retrospectives. *Empirical Software Engineering*, 22(5), pp.2409–2452.

Leveson, N.G. (2017). The Therac-25: 30 Years Later. *Computer*, 50(11), pp.8–11.

Leveson, N.G. and Turner, C.S. (1993). An investigation of the Therac-25 accidents. *Computer*, 26(7), pp.18–41.

Li, H., Liu, Y., Qi, X., Yu, X. and Guo, S. (2023). Structuring meaningful bug-fixing patches to fix software defect. *IET Software*, 17(4), pp.566–581.

Lin, W. and Fan, X. (2009). Software Development Practice for FDA-Compliant Medical Devices. In: *2009 International Joint Conference on Computational Sciences and Optimization*. pp.388–390.

Lindholm, C., Notander, J.P. and Höst, M. (2014). A case study on software risk analysis and planning in medical device development. *Software Quality Journal*, 22(3), pp.469–497.

Lindstrom, L. and Jeffries, R. (2004). Extreme Programming and Agile Software Development Methodologies. *Information Systems Management*, 21(3), pp.41–52.

Liu, B., Zhang, H. and Zhu, S. (2016). An Incremental V-Model Process for Automotive Development. In: *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*. 2016 23rd Asia-Pacific Software Engineering Conference (APSEC). Hamilton, New Zealand: IEEE, pp.225 – 232. Available from: http://ieeexplore.ieee.org/document/7890592/ [accessed 2 Febr uary 2024].

Liu, J.-W., Ho, C.-Y., Chang, J.Y.T. and Tsai, J.C.-A. (2019). The role of Sprint planning and feedback in game development projects: Implications for game quality. *Journal of Systems and Software*, 154, pp.79–91.

Liu, S., Sun, J., Liu, Y., Zhang, Y., Wadhwa, B., Dong, J.S. and Wang, X. (2014). Automatic early defects detection in use case documents. In: *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*. ASE '14: ACM/IEEE International Conference on Automated Software Engineering. Vasteras Sweden: ACM, pp.785–790. Available from: https://dl.acm.org/doi/10.1145/2642937.2642969 [accessed 22 March 2023].

Livermore, J.A. (2008). Factors that Significantly Impact the Implementation of an Agile Software Development Methodology. *Journal of Software*, 3(4), pp.31–36.

Lohan, G., Lang, M. and Conboy, K. (2011). Having a Customer Focus in Agile Software Development. In: Pokorny, J. et al., eds *Information Systems Development*. New York, NY: Springer New York, pp.441–453. Available from: http://link.springer.com/10.1007/978-1-4419-9790-6_35 [accessed 6 January 2025].

Lopes de Souza, P., Lopes de Souza, W. and Ferreira Pires, L. (2021). ScrumOntoBDD: Agile software development based on scrum, ontologies and behaviour-driven development. *Journal of the Brazilian Computer Society*, 27(1), pp.1–45.

Lucassen, G., Dalpiaz, F., van der Werf, J.M.E.M. and Brinkkemper, S. (2016). Improving agile requirements: the Quality User Story framework and tool. *Requirements Engineering*, 21(3), pp.383–403.

Lucassen, G., Dalpiaz, F., Werf, J.M.E.M. van der and Brinkkemper, S. (2016). The Use and Effectiveness of User Stories in Practice. In: Daneva, M. and Pastor, O., eds *Requirements Engineering: Foundation for Software Quality*. Lecture Notes in Computer Science. Cham: Springer International Publishing, pp.205– 222. Available from: http://link.springer.com/10.1007/978-3-319-30282-9_14 [accessed 8 February 2023].

Lui, K.M. and Chan, K.C.C. (2006). Pair programming productivity: Novice–novice vs. expert–expert. *International Journal of Human-Computer Studies*, 64(9), pp.915–925.

Lui, K.M. and Chan, K.C.C. (2004). Test Driven Development and Software Process Improvement in China. In: Eckstein, J. and Baumeister, H., eds *Extreme Programming and Agile Processes in Software Engineering*. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, pp.219 – 222. Available from: http://link.springer.com/10.1007/978-3-540-24853-8_27 [accessed 1 May 2022].

Łukasiewicz, K. and Górski, J. (2018). Introducing agile practices into development processes of safety critical software. In: *Proceedings of the 19th International Conference on Agile Software Development: Companion*. XP '18 Companion: 19th International Conference on Agile Software Development. Porto Portugal: ACM, pp.1–8. Available from: https://dl.acm.org/doi/10.1145/3234152.3234174 [accessed 27 February 2023].

Ma, W., Chen, L., Zhou, Y. and Xu, B. (2016). Do We Have a Chance to Fix Bugs When Refactoring Code Smells? In: *2016 International Conference on Software Analysis, Testing and Evolution (SATE)*. 2016 International Conference on Software Analysis, Testing and Evolution (SATE). pp.24–29.

Maciel, C.L., Machado, P.D.L. and Ramalho, F. (2010). An integration testing approach based on test patterns and MDA techniques. In: *Proceedings of the 8th Latin American Conference*

on Pattern Languages of Programs. SugarLoafPLoP '10: 8th Latin American Conference on Pattern Languages of Programs. Salvador Bahia Brazil: ACM, pp.1–16. Available from: https://dl.acm.org/doi/10.1145/2581507.2581521 [accessed 31 May 2023].

Madeyski, L. (2008). Impact of pair programming on thoroughness and fault detection effectiveness of unit test suites. *Software Process: Improvement and Practice*, 13(3), pp.281–295.

Madeyski, L. and Szała, Ł. (2007). The Impact of Test-Driven Development on Software Development Productivity — An Empirical Study. In: Abrahamsson, P. et al., eds *Software Process Improvement*. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, pp.200–211.

Mahmoud, A.N. and Santos, V. (2021). Statistical Analysis for Revealing Defects in Software Projects: Systematic Literature Review. *International Journal of Advanced Computer Science and Applications* [online], 12(11). Available from: http://thesai.org/Publications/ViewPaper?Volume=12&Issue=11&Code=IJACSA&SerialNo=28 [accessed 27 February 2024].

Makower, J., Meer, A. and Denend, L. (2010). *FDA Impact on US Medical Technology Innovation: A Survey of Over 200 Medical Technology Companies*.

Maksimović, J. and Evtimov, J. (2023). Positivism and post-positivism as the basis of quantitative research in pedagogy. *Research in Pedagogy*, 13(1), pp.208–218.

Malmqvist, J., Hellberg, K., Möllås, G., Rose, R. and Shevlin, M. (2019). Conducting the Pilot Study: A Neglected Part of the Research Process? Methodological Findings Supporting the Importance of Piloting in Qualitative Research Studies. *International Journal of Qualitative Methods*, 18, p.160940691987834.

Malterud, K., Siersma, V.D. and Guassora, A.D. (2016). Sample Size in Qualitative Interview Studies: Guided by Information Power. *Qualitative Health Research*, 26(13), pp.1753–1760.

Mancl, D. and Fraser, S.D. (2021). The Stories We Tell: Experience, Research, or Patterns? In: Gregory, P. and Kruchten, P., eds *Agile Processes in Software Engineering and Extreme Programming – Workshops*. Lecture Notes in Business Information Processing. Cham: Springer International Publishing, pp.215– 221. Available from: https://link.springer.com/10.1007/978-3-030-88583-0_22 [accessed 10 May 2025].

Manjunath, K.N., Jagadeesh, J. and Yogeesh, M. (2013). Achieving quality product in a long term software product development in healthcare application using Lean and Agile principles: Software engineering and software development. In: *2013 International Mutli-Conference on Automation, Computing, Communication, Control and Compressed Sensing (iMac4s)*. Kottayam: IEEE, pp.26–34. Available from: http://ieeexplore.ieee.org/document/6526379/ [accessed 4 December 2021].

Mantyla, M.V., Khomh, F., Adams, B., Engstrom, E. and Petersen, K. (2013). On Rapid Releases and Software Testing. In: *2013 IEEE International Conference on Software Maintenance*. 2013 IEEE International Conference on Software Maintenance (ICSM). Eindhoven, Netherlands: IEEE, pp.20– 29. Available from: http://ieeexplore.ieee.org/document/6676873/ [accessed 15 February 2023].

Mantyla, M.V. and Lassenius, C. (2009). What Types of Defects Are Really Discovered in Code Reviews? *IEEE Transactions on Software Engineering*, 35(3), pp.430–448.

Marbach, P., Rosser, L., Osvalds, G. and Lempia, D. (2015). Principles for Agile Development. *INCOSE International Symposium*, 25(1), pp.524–537.

Marchenko, A., Abrahamsson, P. and Ihme, T. (2009). Long-Term Effects of Test-Driven Development A Case Study. In: Abrahamsson, P., Marchesi, M., and Maurer, F., eds *Agile Processes in Software Engineering and Extreme Programming*. Lecture Notes in Business Information Processing. Berlin, Heidelberg: Springer, pp.13–22.

Mardiah, N., Afifah, F.N., Syahidin, Y. and Sari, I. (2022). Design of an Insurance Patient Visit Information System Using The V-Model Method. *Jurnal E-Komtek (Elektro-Komputer-Teknik)*, 6(2), pp.265–278.

Margarido, I.L., Faria, J.P., Vidal, M. and Vieira, M. (2011). Classification of Defect Types in Requirements Specifications: Literature Review, Proposal and Assessment. In: *Information Systems and Technologies (CISTI)*. 2011 6th Iberian Conference on Information Systems and Technologies (CISTI).

Marshall, B., Cardon, P., Poddar, A. and Fontenot, R. (2013). Does Sample Size Matter in Qualitative Research? A Review of Qualitative Interviews in is Research. *Journal of Computer Information Systems*, 54(1), pp.11–22.

Martin, A., Biddle, R. and Noble, J. (2004). The XP Customer Role in Practice: Three Studies. In: *Agile Development Conference*. Agile Development Conference. Salt Lake City, UT, USA: IEEE, pp.42–54. Available from: http://ieeexplore.ieee.org/document/1359794/ [accessed 28 September 2022].

Martin, R.C. (2014). *Agile software development, principles, patterns, and practices*. First edition, Pearson new international edition. Harlow: Pearson.

Mathers, N., Fox, N. and Hunn, A. (2009). Surveys and Questionnaires., 2009.

Mathur, S. and Malik, S. (2010). Advancements in the V-Model. *International Journal of Computer Applications*, 1(12), pp.30–35.

Matinnejad, R. (2011). Agile Model Driven Development: An Intelligent Compromise. In: *2011 Ninth International Conference on Software Engineering Research, Management and Applications*. 2011 9th International Conference on Software Engineering Research, Management and Applications (SERA). Baltimore, MD, USA: IEEE, pp.197–202. Available from: http://ieeexplore.ieee.org/document/6065641/ [accessed 29 March 2023].

Matthies, C. (2020). Playing with your project data in scrum retrospectives. In: *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings*. ICSE '20: 42nd International Conference on Software Engineering. Seoul South Korea: ACM, pp.113–115. Available from: https://dl.acm.org/doi/10.1145/3377812.3382164 [accessed 19 July 2023].

Maximilien, E.M. and Williams, L. (2003). Assessing test-driven development at IBM. In: *25th International Conference on Software Engineering, 2003. Proceedings.* 25th International

Conference on Software Engineering, 2003. Proceedings. Portland, OR, USA: IEEE, pp.564–569. Available from: http://ieeexplore.ieee.org/document/1201238/ [accessed 1 May 2022].

McCaffery, F., Lepmets, M., Trektere, K., Ozcan Top, O. and Pikkarainen, M. (2016). Agile medical device software development: Introducing agile practices into MDevSPICE., 8, pp.133–142.

McCaffery, F., Casey, V., Sivakumar, M.S., Coleman, G., Donnelly, P. and Burton, J. (2012). Medical Device Software Traceability. In: Cleland-Huang, J., Gotel, O., and Zisman, A., eds *Software and Systems Traceability*. London: Springer London, pp.321–339. Available from: http://link.springer.com/10.1007/978-1-4471-2239-5_15 [accessed 8 October 2025].

McCaffery, F., Trektere, K. and Ozcan-Top, O. (2016). Agile – Is it Suitable for Medical Device Software Development? In: Clarke, P. M. et al., eds *Software Process Improvement and Capability Determination*. Communications in Computer and Information Science. Cham: Springer International Publishing, pp.417–422.

McDougall, R. (2015). Reviewing Literature in Bioethics Research: Increasing Rigour in Non-Systematic Reviews. *Bioethics*, 29(7), pp.523–528.

McGee, E.T. and McGregor, J.D. (2016). Using Dynamic Adaptive Systems in Safety-Critical Domains. In: *2016 IEEE/ACM 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. 2016 IEEE/ACM 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS). pp.115–121. Available from: https://ieeexplore.ieee.org/document/7830554.

McHugh, M., Cawley, O., McCaffery, F., Richardson, I. and Wang, X. (2013). An agile V-model for medical device software development to overcome the challenges with plan-driven software development lifecycles. In: *2013 5th International Workshop on Software Engineering in Health Care (SEHC)*. pp.12 – 19. Available from: https://ieeexplore.ieee.org/document/6602471 [accessed 22 April 2025].

McHugh, M., McCaffery, F., Fitzgerald, B., Stol, K.-J., Casey, V. and Coady, G. (2013). Balancing Agility and Discipline in a Medical Device Software Organisation. In: Woronowicz, T. et al., eds *Software Process Improvement and Capability Determination*. Communications in Computer and Information Science. Berlin, Heidelberg: Springer Berlin Heidelberg, pp.199–210. Available from: http://link.springer.com/10.1007/978-3-642-38833-0_18 [accessed 28 October 2021].

McHugh, M., McCaffery, F., Casey, V. and Pikkarainen, M. (2012). Integrating Agile Practices with a Medical Device Software Development Lifecycle., 2012.

McHugh, M., McCaffery, F. and Casey, V. (2012). Barriers to Adopting Agile Practices When Developing Medical Device Software. In: Mas, A. et al., eds *Software Process Improvement and Capability Determination*. Communications in Computer and Information Science. Berlin, Heidelberg: Springer Berlin Heidelberg, pp.141 – 147. Available from: http://link.springer.com/10.1007/978-3-642-30439-2_13 [accessed 28 October 2021].

McHugh, M., McCaffery, F. and Casey, V. (2011). Standalone Software as an Active Medical Device. In: pp.97–107.

McHugh, M., McCaffery, F. and Coady, G. (2015). Adopting Agile Practices when Developing Medical Device Software. *Computer Engineering & Information Technology* [online], 04(02). Available from: http://scitechnol.com/adopting-agile-practices-when-developing-medical-device-software-3Pbz.php?article_id=3461 [accessed 20 June 2023].

McHugh, M., McCaffery, F. and Coady, G. (2014). An Agile Implementation within a Medical Device Software Organisation. In: Mitasiunas, A. et al., eds *Software Process Improvement and Capability Determination*. Communications in Computer and Information Science. Cham: Springer International Publishing, pp.190– 201. Available from: http://link.springer.com/10.1007/978-3-319-13036-1_17 [accessed 31 January 2024].

McIntosh, S., Kamei, Y., Adams, B. and Hassan, A.E. (2016). An empirical study of the impact of modern code review practices on software quality. *Empirical Software Engineering*, 21(5), pp.2146–2189.

McIntosh, S., Kamei, Y., Adams, B. and Hassan, A.E. (2014). The impact of code review coverage and code review participation on software quality: a case study of the qt, VTK, and ITK projects. In: *Proceedings of the 11th Working Conference on Mining Software Repositories - MSR 2014*. the 11th Working Conference. Hyderabad, India: ACM Press, pp.192–201. Available from: http://dl.acm.org/citation.cfm?doid=2597073.2597076 [accessed 17 August 2022].

MDCG. (2019). MDCG 2019-11 Guidance on Qualification and Classification of Software in Regulation (EU) 2017/745 – MDR and Regulation (EU) 2017/746 – IVDR., 2019. Available from: https://ec.europa.eu/health/system/files/2020-09/md_mdcg_2019_11_guidance_qualification_classification_software_en_0.pdf [accessed 2 June 2022].

Medeiros, J., Vasconcelos, A., Goulão, M., Silva, C. and Araújo, J. (2017). An approach based on design practices to specify requirements in agile projects. In: *Proceedings of the Symposium on Applied Computing*. SAC 2017: Symposium on Applied Computing. Marrakech Morocco: ACM, pp.1114–1121. Available from: https://dl.acm.org/doi/10.1145/3019612.3019753 [accessed 1 February 2023].

Mehrfard, H. and Hamou-Lhadj, A. (2011). The Impact of Regulatory Compliance on Agile Software Processes with a Focus on the FDA Guidelines for Medical Device Software. *International Journal of Information System Modeling and Design (IJISMD)*, 2(2), pp.67–81.

Meier, A. and Ivarsson, J.C. (2013). Agile Software Development and Service Science., 3(3).

Mellegard, N., Burden, H., Levin, D., Lind, K. and Magazinius, A. (2020). Contrasting Big Bang with Continuous Integration Through Defect Reports. *IEEE Software*, 37(3), pp.14–20.

Melvin, T. (2022). The European Medical Device Regulation–What Biomedical Engineers Need to Know. *IEEE Journal of Translational Engineering in Health and Medicine*, 10, p.4800105.

Memon, M.A., Baloch, M.-U.-R.M., Memon, M. and Musavi, S.H.A. (2018). Defects Prediction and Prevention Approaches for Quality Software Development. *International Journal of Advanced Computer Science and Applications (ijacsa)* [online], 9(8). Available

from: https://thesai.org/Publications/ViewPaper?Volume=9&Issue=8&Code=ijacsa&SerialNo=57 [accessed 24 February 2024].

Michael, K.A. and Boniface, K.A. (2014). Inadequate Requirements Engineering Process: A Key Factor for Poor Software Development in Developing Nations: A Case Study., 8(9).

Middleton, P. and Joyce, D. (2012). Lean Software Management: BBC Worldwide Case Study. *IEEE Transactions on Engineering Management*, 59(1), pp.20–32.

Mili, A., Cukic, B., Xia, T. and Ben Ayed, R. (1999). Combining fault avoidance, fault removal and fault tolerance: an integrated model. In: *14th IEEE International Conference on Automated Software Engineering*. 14th IEEE International Conference on Automated Software Engineering. pp.137–146.

Miller, R.W. and Collins, C.T. (2002). Acceptance Testing., 2002.

Mockus, A., Nagappan, N. and Dinh-Trong, T.T. (2009). Test coverage and post-verification defects: A multiple case study. In: *2009 3rd International Symposium on Empirical Software Engineering and Measurement*. 2009 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM). Lake Buena Vista, FL, USA: IEEE, pp.291–301. Available from: http://ieeexplore.ieee.org/document/5315981/ [accessed 22 July 2022].

Moe, M.M. and Oo, K.K. (2020). Evaluation of Quality, Productivity, and Defect by applying Test-Driven Development to perform Unit Tests. In: *2020 IEEE 9th Global Conference on Consumer Electronics (GCCE)*. 2020 IEEE 9th Global Conference on Consumer Electronics (GCCE). pp.435–436.

Mooghali, M., Ross, J.S., Kadakia, K.T. and Dhruva, S.S. (2023). Characterization of US Food and Drug Administration Class I Recalls from 2018 to 2022 for Moderate- and High-Risk Medical Devices: A Cross-Sectional Study. *Medical Devices: Evidence and Research*, Volume 16, pp.111–122.

Morales, R., McIntosh, S. and Khomh, F. (2015). Do code review practices impact design quality? A case study of the Qt, VTK, and ITK projects. In: *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. 2015 IEEE 22nd International Conference on Software Analysis, Evolution and Reengineering (SANER). Montreal, QC, Canada: IEEE, pp.171 – 180. Available from: http://ieeexplore.ieee.org/document/7081827/ [accessed 23 August 2022].

Moser, A. and Korstjens, I. (2018). Series: Practical guidance to qualitative research. Part 3: Sampling, data collection and analysis. *European Journal of General Practice*, 24(1), pp.9–18.

Moser, R., Abrahamsson, P., Pedrycz, W., Sillitti, A. and Succi, G. (2008). A Case Study on the Impact of Refactoring on Quality and Productivity in an Agile Team. In: Meyer, B., Nawrocki, J. R., and Walter, B., eds *Balancing Agility and Formalism in Software Engineering*. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, pp.252–266. Available from: http://link.springer.com/10.1007/978-3-540-85279-7_20 [accessed 25 July 2022].

Mucambe, B., Tereso, A., Faria, J. and Mateus, T. (2020). Large-Scale Agile Frameworks: Dealing With Interdependences., 2020.

Munassar, N.M.A. and Govardhan, A. (2010). A Comparison Between Five Models Of Software Engineering., 7(5).

Munir, H., Wnuk, K., Petersen, K. and Moayyed, M. (2014). An experimental evaluation of test driven development vs. test-last development with industry professionals. In: *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering - EASE '14*. the 18th International Conference. London, England, United Kingdom: ACM Press, pp.1–10. Available from: http://dl.acm.org/citation.cfm?doid=2601248.2601267 [accessed 15 June 2022].

Myklebust, T., Lyngby, N. and Stålhane, T. (2018). Agile practices when developing safety systems. *Los Angeles*, 2018.

Myklebust, T., Stålhane, T. and Hanssen, G. (2016). Use of Agile Practices when developing Safety-Critical software.

Nagappan, N., Maximilien, E.M., Bhat, T. and Williams, L. (2008). Realizing quality improvement through test driven development: results and experiences of four industrial teams. *Empirical Software Engineering*, 13(3), pp.289–302.

Nakkeeran, N. (2016). Is Sampling a Misnomer in Qualitative Research? *Sociological Bulletin*, 65(1), pp.40–49.

Nazia, S. (2019). Awareness Regarding Importance of SDLC among IT Professionals and Students: A Survey. *International Journal of Computer Applications*, 178(18), pp.38–46.

Neumann, M. (2021). The Integrated List of Agile Practices - A Tertiary Study., 17 November 2021.

Nezhad, A.S., Lukkien, J.J. and Mak, Rudolf.H. (2018). Behavior-driven Development for Real-time Embedded Systems. In: *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA). Turin: IEEE, pp.59–66. Available from: https://ieeexplore.ieee.org/document/8502653/ [accessed 1 February 2023].

Nielsen, P.A. and Heeager, L.T. (2017). The dynamics of agile practices for safety-critical software development. In: *Proceedings of the XP2017 Scientific Workshops*. Cologne Germany: ACM, pp.1–6. Available from: https://dl.acm.org/doi/10.1145/3120459.3120481 [accessed 20 December 2021].

Noll, J., Razzak, M.A., Bass, J.M. and Beecham, S. (2017). A Study of the Scrum Master's Role. In: Felderer, M. et al., eds *Product-Focused Software Process Improvement*. Lecture Notes in Computer Science. Cham: Springer International Publishing, pp.307–323. Available from: http://link.springer.com/10.1007/978-3-319-69926-4_22 [accessed 30 December 2024].

Noor, R. and Fahad Khan, M. (2014). Defect Management in Agile Software Development. *International Journal of Modern Education and Computer Science*, 6(3), pp.55–60.

Nüssler, A. (2023). The new European Medical Device Regulation: Friend or foe for hospitals and patients? *Injury*, 54, p.110907.

Ochodek, M., Hebig, R., Meding, W., Frost, G. and Staron, M. (2020). Recognizing lines of code violating company-specific coding guidelines using machine learning: A Method and Its Evaluation. *Empirical Software Engineering*, 25(1), pp.220–265.

Oliveira Lopes, M.V.D. (2012). About case studies and experience reports. *Rev Rene*, 13(4), pp.1–1.

Olmsted, J. (2024). Research Reliability and Validity: Why do they matter? *Journal of dental hygiene: JDH*, 98(6), pp.53–57.

Orellana, G., Laghari, G., Murgia, A. and Demeyer, S. (2017). On the Differences between Unit and Integration Testing in the TravisTorrent Dataset. In: *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR). Buenos Aires, Argentina: IEEE, pp.451–454. Available from: http://ieeexplore.ieee.org/document/7962394/ [accessed 18 July 2022].

Orr, K. (2001). Chapter 23. Adaptive Software Development., 2001.

Ouni, A., Kessentini, M., Sahraoui, H., Inoue, K. and Deb, K. (2016). Multi-Criteria Code Refactoring Using Search-Based Software Engineering: An Industrial Case Study. *ACM Transactions on Software Engineering and Methodology*, 25(3), p.23:1-23:53.

Ouni, A., Kessentini, M. and Sahraoui, H. (2013). Search-Based Refactoring Using Recorded Code Changes. In: *2013 17th European Conference on Software Maintenance and Reengineering*. 2013 17th European Conference on Software Maintenance and Reengineering (CSMR 2013). Genova: IEEE, pp.221 – 230. Available from: http://ieeexplore.ieee.org/document/6498470/ [accessed 11 August 2022].

Özcan-Top, Özden and McCaffery, F. (2018a). A hybrid assessment approach for medical device software development companies. *Journal of Software: Evolution and Process*, 30(7), p.e1929.

Özcan-Top, Özden and McCaffery, F. (2018b). A hybrid assessment approach for medical device software development companies. *Journal of Software: Evolution and Process*, 30(7), p.e1929.

Özcan-Top, Ö and McCaffery, F. (2018). Conformance to Medical Device Software Development Requirements with XP and Scrum Implementation., 2018, p.8.

Özcan-Top, Ö. and McCaffery, F. (2017). How Does Scrum Conform to the Regulatory Requirements Defined in MDevSPICE®? In: Mas, A. et al., eds *Software Process Improvement and Capability Determination*. Communications in Computer and Information Science. Cham: Springer International Publishing, pp.257–268. Available from: http://link.springer.com/10.1007/978-3-319-67383-7_19 [accessed 7 January 2022].

Özçelik, O. and Altilar, D.T. (2020). Test-Driven Approach for Safety-Critical Software Development. *Journal of Software*, July 2020, pp.904–911.

Padmanabhuni, V.V.K., Tadiparthi, H.P., Yanamadala, M. and Madina, S. (2012). Effective Pair Programming Practice- An Experimental Study., 3(4), p.9.

Paige, R.F., Chivers, H., McDermid, J.A. and Stephenson, Z.R. (2005). High-integrity extreme programming. In: *Proceedings of the 2005 ACM symposium on Applied computing*. SAC '05. New York, NY, USA: Association for Computing Machinery, pp.1518–1523. Available from: https://doi.org/10.1145/1066677.1067019 [accessed 5 January 2025].

Palinkas, L.A., Horwitz, S.M., Green, C.A., Wisdom, J.P., Duan, N. and Hoagwood, K. (2015). Purposeful Sampling for Qualitative Data Collection and Analysis in Mixed Method Implementation Research. *Administration and Policy in Mental Health and Mental Health Services Research*, 42(5), pp.533–544.

Palmer, S.R. and Felsing, M. (2001). *A Practical Guide to Feature-Driven Development*. 1st edn Pearson Education.

Palomba, F., Zaidman, A., Oliveto, R. and De Lucia, A. (2017). An Exploratory Study on the Relationship between Changes and Refactoring. In: *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*. 2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC). Buenos Aires, Argentina: IEEE, pp.176–185. Available from: http://ieeexplore.ieee.org/document/7961515/ [accessed 2 August 2022].

Pančur, M. and Ciglarič, M. (2011). Impact of test-driven development on productivity, code and tests: A controlled experiment. *Information and Software Technology*, 53(6), pp.557–573.

Pandey, A.K. (2019). Defect, Failure, Fault, Error and Problem. *Journal of Information and Computational Science*, 9(7), p.11.

Pandit, P. and Tahiliani, S. (2015). AgileUAT: A Framework for User Acceptance Testing based on User Stories and Acceptance Criteria. *International Journal of Computer Applications*, 120(10), pp.16–21.

Panhwar, A.H., Ansari, S. and Shah, A.A. (2017). Post-positivism: an effective paradigm for social and educational research. *International Research Journal of Arts and Humanities*, 45(45), pp.253–259.

Papis, B., Grochowski, K., Subzda, K. and Sijko, K. (2022). Experimental Evaluation of Test-Driven Development with Interns Working on a Real Industrial Project. *IEEE Transactions on Software Engineering*, 48(5), pp.1644–1664.

Park, Y.S., Konge, L. and Artino, A.R. (2020). The Positivism Paradigm of Research. *Academic Medicine*, 95(5), pp.690–694.

Parsons, D., Ryu, H. and Lal, R. (2008). Better, Not More Expensive, Faster? The Perceived Effects of Pair Programming in Survey Data., 2008, p.10.

Pascarella, L., Spadini, D., Palomba, F. and Bacchelli, A. (2019). On The Effect of Code Review on Code Smells., 20 December 2019. Available from: http://arxiv.org/abs/1912.10098 [accessed 1 September 2022].

Pashkov, V., Gutorova, N. and Harkusha, A. (2016). Medical device software: defining key terms., January 2016, p.6.

Pashkov, V. and Harkusha, A. (2020). Stand-Alone Software as a Medical Device: Qualification and Liability Issues. *Wiadomości lekarskie (Warsaw, Poland: 1960)*, LXXIII, p.2282.

Patel, M. and Patel, N. (2019). Exploring Research Methodology: Review Article., 2019, p.8.

Paul, R., Turzo, A.K. and Bosu, A. (2021). Why Security Defects Go Unnoticed During Code Reviews? A Case-Control Study of the Chromium OS Project. In: *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). Madrid, ES: IEEE, pp.1373–1385. Available from: https://ieeexplore.ieee.org/document/9402130/ [accessed 5 September 2022].

Peffers, K., Tuunanen, T., Rothenberger, M.A. and Chatterjee, S. (2007). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, 24(3), pp.45–77.

Péraire, C., Edwards, M., Fernandes, A., Mancin, E. and Carroll, K. (2007). The IBM Rational Unified Process for System z., July 2007.

Perkusich, M., Silva, Ana, Silva, André, Araújo, T., Barbosa, R., Ramos, F., Costa, A. and Dilorenzo, E. (2017). Ordering the Product Backlog in Agile Software Development Projects: A Systematic Literature Review.

Petersen, K., Wohlin, C. and Baca, D. (2009). The Waterfall Model in Large-Scale Development. In: Bomarius, F. et al., eds *Product-Focused Software Process Improvement*. Lecture Notes in Business Information Processing. Berlin, Heidelberg: Springer Berlin Heidelberg, pp.386–400. Available from: http://link.springer.com/10.1007/978-3-642-02152-7_29 [accessed 25 December 2024].

Petrić, J., Hall, T. and Bowes, D. (2020). Which Software Faults Are Tests Not Detecting? In: *Proceedings of the Evaluation and Assessment in Software Engineering*. EASE '20: Evaluation and Assessment in Software Engineering. Trondheim Norway: ACM, pp.160–169. Available from: https://dl.acm.org/doi/10.1145/3383219.3383236 [accessed 21 July 2022].

Phaphoom, N., Sillitti, A. and Succi, G. (2011). Pair Programming and Software Defects – An Industrial Case Study. In: Sillitti, A. et al., eds *Agile Processes in Software Engineering and Extreme Programming*. Lecture Notes in Business Information Processing. Berlin, Heidelberg: Springer Berlin Heidelberg, pp.208 – 222. Available from: http://link.springer.com/10.1007/978-3-642-20677-1_15 [accessed 17 May 2022].

Phongpaibul, M. and Boehm, B. (2007). A Replicate Empirical Comparison between Pair Development and Software Development with Inspection. In: *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*. First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007). pp.265–274.

Phongpaibul, M. and Boehm, B. (2006). An empirical comparison between pair development and software inspection in Thailand. In: *Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering - ISESE '06*. The 2006 ACM/IEEE internatio nal symposium. Rio de Janeiro, Brazil: ACM Press, p.85. Available from: http://portal.acm.org/citation.cfm?doid=1159733.1159749 [accessed 10 May 2022].

Pieterse, V. (2008). Reflections on coding standards in tertiary Computer Science education: festschrift: dedicated to Derrick Kourie. *South African Computer Journal*, 2008(41), pp.29–37.

Pinto, G., Castor, F., Bonifacio, R. and Rebouças, M. (2018). Work practices and challenges in continuous integration: A survey with Travis CI users: Work practices and challenges: A survey with Travis CI users. *Software: Practice and Experience*, 48(12), pp.2223–2236.

Ponto, J. (2015). Understanding and Evaluating Survey Research. *Journal of the Advanced Practitioner in Oncology* [online], 6(2). Available from: http://www.advancedpractitioner.com/issues/volume-6,-number-2-(marapr-2015)/understanding-and-evaluating-survey-research.aspx [accessed 18 November 2022].

Popic, S., Velikic, G., Jaroslav, H., Spasic, Z. and Vulic, M. (2018). The Benefits of the Coding Standards Enforcement and its Influence on the Developers' Coding Behaviour: A Case Study on Two Small Projects. In: *2018 26th Telecommunications Forum (TELFOR)*. 2018 26th Telecommunications Forum (TELFOR). Belgrade: IEEE, pp.420–425. Available from: https://ieeexplore.ieee.org/document/8612149/ [accessed 22 March 2023].

Poppendieck, M. and Poppendieck, T. (2010). *Lean software development: an agile toolkit*. Nachdr. Boston: Addison-Wesley.

Priyadarshini, A. (2020). *Conducting and Analysing Semi-Structured Interviews: A Study of Open Innovation in Food Firms in Ireland* [online]. 1 Oliver's Yard, 55 City Road, London EC1Y 1SP United Kingdom: SAGE Publications Ltd. Available from: https://methods.sagepub.com/case/semi-structured-interviews-study-of-open-innovation-in-food-firms-ireland [accessed 12 September 2024].

Pruim, R., Gîrjău, M.-C. and Horton, N.J. (2022). The importance of good coding practices for data scientists., 8 October 2022. Available from: http://arxiv.org/abs/2210.03991 [accessed 22 March 2023].

Putta, A., Paasivaara, M. and Lassenius, C. (2018). Benefits and Challenges of Adopting the Scaled Agile Framework (SAFe): Preliminary Results from a Multivocal Literature Review. In: Kuhrmann, M. et al., eds *Product-Focused Software Process Improvement*. Cham: Springer International Publishing, pp.334–351.

Rahman, A., Agrawal, A., Krishna, R. and Sobran, A. (2018). Characterizing The Influence of Continuous Integration. Empirical Results from 250+ Open Source and Proprietary Projects. In: *Proceedings of the 4th ACM SIGSOFT International Workshop on Software Analytics*. pp.8–14. Available from: http://arxiv.org/abs/1711.03933 [accessed 20 September 2022].

Rahy, S. and Bass, J.M. (2022). Managing non-functional requirements in agile software development. *IET Software*, 16(1), pp.60–72.

Rajaram, H.K., Loane, J., MacMahon, S.T. and McCaffery, F. (2019a). A Framework for Taxonomy Based Testing Using Classification of Defects in Health Software-SW91. In: Walker, A., O'Connor, R. V., and Messnarz, R., eds *Systems, Software and Services Process Improvement*. Communications in Computer and Information Science. Cham: Springer International Publishing, pp.606–618. Available from: http://link.springer.com/10.1007/978-3-030-28005-5_47 [accessed 16 February 2022].

Rajaram, H.K., Loane, J., MacMahon, S.T. and McCaffery, F. (2019b). Taxonomy-based testing and validation of a new defect classification for health software. *Journal of Software: Evolution and Process*, 31(1), p.e1985.

Ramdoo, V.D. and Huzooree, G. (2015). Strategies to Reduce Rework in Software Development on an Organisation in Mauritius. *International Journal of Software Engineering & Applications*, 6(5), pp.09–20.

Ramessur, M.A. and Nagowah, S.D. (2020). Factors Affecting Sprint Effort Estimation. In: Pati, B. et al., eds *Advanced Computing and Intelligent Engineering*. Advances in Intelligent Systems and Computing. Singapore: Springer Singapore, pp.507–518. Available from: http://link.springer.com/10.1007/978-981-15-1483-8_43 [accessed 26 March 2023].

Ramler, R., Wolfmaier, K. and Kopetzky, T. (2013). A Replicated Study on Random Test Case Generation and Manual Unit Testing: How Many Bugs Do Professional Developers Find? In: *2013 IEEE 37th Annual Computer Software and Applications Conference*. 2013 IEEE 37th Annual Computer Software and Applications Conference (COMPSAC). Kyoto, Japan: IEEE, pp.484–491. Available from: https://ieeexplore.ieee.org/document/6649872 [accessed 11 July 2022].

Ramsberg, N. (2018). 3.0 Research methodology. *Leading design* [online], 3 April 2018. Available from: https://medium.com/ma-leading-design/research-methodology-45871a6e4cc7 [accessed 9 November 2022].

Rashid, J., Mahmood, T. and Nisar, M.W. (2019). A Study on Software Metrics and its Impact on Software Quality., 24(1).

Rasmussen, R., Hughes, T., Jenks, J.R. and Skach, J. (2009). Adopting Agile in an FDA Regulated Environment. In: *2009 Agile Conference*. pp.151–155.

Ratzinger, J., Sigmund, T. and Gall, H.C. (2008). On the relation of refactorings and software defect prediction. In: *Proceedings of the 2008 international working conference on Mining software repositories*. MSR '08. New York, NY, USA: Association for Computing Machinery, pp.35–38. Available from: https://doi.org/10.1145/1370750.1370759 [accessed 27 July 2022].

Rawat, M.S. and Dubey, S.K. (2012). Software Defect Prediction Models for Quality Improvement: A Literature Study., 9(5).

Rayudu, D.M. and Kumar, V. (2014). The Impact of Test-Driven Development on Software Defects and Cost: A Comparative Case Study., 2014, p.10.

Rech, J. (2007). Handling of Software Quality Defects in Agile Software Development. In: Stamelos, I. G. and Sfetsos, P., eds *Agile Software Development Quality Assurance*. IGI Global, pp.90–113. Available from: http://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/978-1-59904-216-9.ch005 [accessed 18 March 2022].

Reddy, K.N. and Rao, A.A. (2009). A Quantitative Evaluation of Software Quality Enhancement by Refactoring Using Dependency Oriented Complexity Metrics. In: *2009 Second International Conference on Emerging Trends in Engineering & Technology*. 2009 Second International Conference on Emerging Trends in Engineering & Technology. Nagpur,

India: IEEE, pp.1011–1018. Available from: http://ieeexplore.ieee.org/document/5395508/ [accessed 27 July 2022].

Reeves, S., Peller, J., Goldman, J. and Kitto, S. (2013). Ethnography in qualitative educational research: AMEE Guide No. 80. *Medical Teacher*, 35(8), pp.e1365–e1379.

Rigby, P.C. and Bird, C. (2013). Convergent contemporary software peer review practices. In: *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2013*. the 2013 9th Joint Meeting. Saint Petersburg, Russia: ACM Press, p.202. Available from: http://dl.acm.org/citation.cfm?doid=2491411.2491444 [accessed 23 August 2022].

Robinson, R.S. (2014). Purposive Sampling. In: Michalos, A. C., ed. *Encyclopaedia of Quality of Life and Well-Being Research*. Dordrecht: Springer Netherlands, pp.5243–5245. Available from: https://doi.org/10.1007/978-94-007-0753-5_2337 [accessed 26 June 2023].

Rodeghero, P., Jiang, S., Armaly, A. and McMillan, C. (2017). Detecting User Story Information in Developer-Client Conversations to Generate Extractive Summaries. In: *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE). Buenos Aires: IEEE, pp.49–59. Available from: http://ieeexplore.ieee.org/document/7985649/ [accessed 8 February 2023].

Ronquillo, J.G. and Zuckerman, D.M. (2017). Software-Related Recalls of Health Information Technology and Other Medical Devices: Implications for FDA Regulation of Digital Health. *The Milbank Quarterly*, 95(3), pp.535–553.

Rottier, P. and Rodrigues, V. (2008). Agile Development in a Medical Device Company. In: pp.218–223.

Roulston, K. (2023). Interviews in Qualitative Research. In: *The Encyclopaedia of Applied Linguistics*. John Wiley & Sons, Ltd, pp.1 – 9. Available from: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781405198431.wbeal0572.pub3 [accessed 17 October 2024].

Royce, W.W. (1987). Managing the development of large software systems: concepts and techniques. In: *Proceedings of the 9th international conference on Software Engineering*. ICSE '87. Washington, DC, USA: IEEE Computer Society Press, pp.328–338.

Rubin, K.S. (2012). *Essential Scrum: A Practical Guide to the Most Popular Agile Process*. Addison-Wesley Professional.

Runeson, P. and Andrews, A. (2003). Detection or isolation of defects? An experimental comparison of unit testing and code inspection. In: *14th International Symposium on Software Reliability Engineering, 2003. ISSRE 2003.* 14th International Symposium on Software Reliability Engineering, 2003. ISSRE 2003. pp.3–13.

Ruparelia, N.B. (2010). Software development lifecycle models. *ACM SIGSOFT Software Engineering Notes*, 35(3), pp.8–13.

Russo, B., Scotto, M., Sillitti, A. and Succi, G. (2010). *Agile Technologies in Open Source Development:* [online]. IGI Global. Available from: http://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/978-1-59904-681-5 [accessed 13 February 2023].

Rust, A., Bishop, B. and McDaid, K. (2007). Investigating the Potential of Test-Driven Development for Spreadsheet Engineering., 2007.

Ryan, A.B. (2006). Post-positivism Approaches to Research., 2006, p.15.

Sabane, A., Di Penta, M., Antoniol, G. and Gueheneuc, Y. (2013). A Study on the Relation between Antipatterns and the Cost of Class Unit Testing. In: *2013 17th European Conference on Software Maintenance and Reengineering*. 2013 17th European Conference on Software Maintenance and Reengineering (CSMR 2013). Genova: IEEE, pp.167–176. Available from: http://ieeexplore.ieee.org/document/6498465/ [accessed 22 July 2022].

Sanchez, J., Williams, L. and Maximilien, E. (2008). A Longitudinal Study of the Use of a Test-Driven Development Practice in Industry., 1 July 2008.

Sanchez, J.C., Williams, L. and Maximilien, E.M. (2007). On the Sustained Use of a Test-Driven Development Practice at IBM. In: *Agile 2007 (AGILE 2007)*. Agile 2007. Washington, DC: IEEE, pp.5–14. Available from: http://ieeexplore.ieee.org/document/4293571/ [accessed 2 May 2022].

Saravanos, A. and Curinga, M.X. (2023). Simulating the Software Development Lifecycle: The Waterfall Model. *Applied System Innovation*, 6(6), p.108.

Saunders, M., Lewis, P. and Thornhill, A. (2019). *Research methods for business students*. Eighth edition. Harlow, England: Pearson.

Scanniello, G., Romano, S., Fucci, D., Turhan, B. and Juristo, N. (2016). Students' and professionals' perceptions of test-driven development: a focus group study. In: *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. SAC '16. New York, NY, USA: Association for Computing Machinery, pp.1422 – 1427. Available from: https://doi.org/10.1145/2851613.2851778 [accessed 27 June 2022].

Schmidt, C., Kardorff, E. von and Steinke, I., eds. (2004). *A companion to qualitative research*. London; Thousand Oaks, Calif: Sage Publications.

Schneider, A.E. and Müller-Sägebrecht, S. (2024). Why Medical Devices Fail and How Companies Learn from Them. An Empirical Exploration of Recall Data., June 2024.

Schwaber, K. (1997). SCRUM Development Process. In: Sutherland, J. et al., eds *Business Object Design and Implementation*. London: Springer London, pp.117–134. Available from: http://link.springer.com/10.1007/978-1-4471-0947-1_11 [accessed 30 December 2024].

Schwaber, K. and Sutherland, J. (2020). 2020-Scrum-Guide-US., November 2020. Available from: https://www.scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf [accessed 30 December 2024].

Scott, E., Tõemets, T. and Pfahl, D. (2021). An Empirical Study of User Story Quality and Its Impact on Open Source Project Performance. In: Winkler, D. et al., eds *Software Quality: Future Perspectives on Software Engineering Quality*. Lecture Notes in Business Information Processing. Cham: Springer International Publishing, pp.119–138.

Scott, H. and Wohlin, C. (2008). Capture-recapture in software unit testing: a case study. In: *Proceedings of the Second ACM-IEEE international symposium on Empirical software*

engineering and measurement. ESEM '08. New York, NY, USA: Association for Computing Machinery, pp.32–40. Available from: https://doi.org/10.1145/1414004.1414012 [accessed 22 July 2022].

Scrum Alliance. (2024). Core-Scrum., 2024. Available from: https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.scrumalliance.org%2FScrumRedesignDEVSite%2Fmedia%2FScrumAllianceMedia%2FFiles%2520and%2520PDFs%2FLearn%2520About%2520Scrum%2FCore-Scrum.pdf&psig=AOvVaw1AipBh6JiZzX4KSfLr3fBF&ust=1735678428177000&source=images&cd=vfe&opi=89978449&ved=0CAYQrpoMahcKEwjYupOTsNCKAxUAAAAAHQAAAAAQBg.

Sedano, T., Ralph, P. and Peraire, C. (2019). The Product Backlog. In: *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). Montreal, QC, Canada: IEEE, pp.200–211. Available from: https://ieeexplore.ieee.org/document/8812076/ [accessed 23 March 2023].

Sedgwick Brand Protection. (2022a). *EU State of the nation 2022 recall index report*. Available from: https://marketing.sedgwick.com/acton/media/4952/2022-state-of-the-nation-recall-index-report.

Sedgwick Brand Protection. (2024). *European Recall Index - 2024 Edition 3* [online]. Available from: https://marketing.sedgwick.com/acton/media/4952/European-2024-recall-index-report-download-and-webinar-edition-3 [accessed 28 November 2024].

Sedgwick Brand Protection. (2026a). *Sedgwick Brand Protection - Q4 2025 Recall Index Report - European Edition*. Available from: https://marketing.sedgwick.com/acton/media/4952/2026-european-recall-index-report-download-us-cross-promo.

Sedgwick Brand Protection. (2026b). *Sedgwick Brand Protection - Q4 2025 Recall Index Report - US Edition*. Available from: https://marketing.sedgwick.com/acton/media/4952/2026-us-product-safety-and-recall-index-report-eu-cross-promo.

Sedgwick Brand Protection. (2022b). *State of the Nation 2022 Recall Index Report* [online]. Available from: https://marketing.sedgwick.com/acton/media/4952/2022-state-of-the-nation-recall-index-report [accessed 6 September 2022].

Sein, Henfridsson, Purao, Rossi, and Lindgren. (2011). Action Design Research. *MIS Quarterly*, 35(1), p.37.

Senapathi, M., Buchan, J. and Osman, H. (2019). DevOps Capabilities, Practices, and Challenges: Insights from a Case Study., 2019.

Senarath, S.U. (2021). Waterfall Methodology, Prototyping and Agile Development., 2021. Available from: http://rgdoi.net/10.13140/RG.2.2.17918.72001 [accessed 26 December 2024].

Serra, D., Grano, G., Palomba, F., Ferrucci, F., Gall, H.C. and Bacchelli, A. (2019). On the Ef fectiveness of Manual and Automatic Unit Test Generation: Ten Years Later. In: *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR). pp.121–125.

Seshu Kumar, A.N. and Vasavi, S. (2012). Effective Unit Testing Framework for Automation of Windows Applications. In: Kumar M., A., R., S., and Kumar, T. V. S., eds *Proceedings of International Conference on Advances in Computing*. Advances in Intelligent Systems and Computing. New Delhi: Springer India, pp.813–822.

Shimagaki, J., Kamei, Y., McIntosh, S., Hassan, A.E. and Ubayashi, N. (2016). A study of the quality-impacting practices of modern code review at Sony mobile. In: *Proceedings of the 38th International Conference on Software Engineering Companion*. ICSE '16: 38th International Conference on Software Engineering. Austin Texas: ACM, pp.212–221. Available from: https://dl.acm.org/doi/10.1145/2889160.2889243 [accessed 17 August 2022].

Shroff, V., Reid, L. and Richardson, I. (2011). A Proposed Framework for Software Quality in the Healthcare and Medical Industry., 2011.

Shukla, A. and Williams, L. (2002). Adapting extreme programming for a core software engineering course. In: *Proceedings 15th Conference on Software Engineering Education and Training (CSEE&T 2002)*. CSEE&T 2002. 15th Conference on Software Engineering Education and Training. Covington, KY, USA: IEEE Comput. Soc, pp.184–191. Available from: http://ieeexplore.ieee.org/document/995210/ [accessed 13 February 2023].

Simone, L.K. (2013). Software-Related Recalls: An Analysis of Records. *Biomedical Instrumentation & Technology*, 47(6), pp.514–522.

Singh, R. and Akhtar Khan, I. (2012). An Approach for Integration Testing in Online Retail Applications. *International Journal of Computer Science and Information Technology*, 4(3), pp.141–158.

Sirshar, M., Daud, M., Noor, J. and Rashid, R. (2019). A Framework for Software Defect Management Process in Software Quality Assurance., December 2019, p.8.

Sison, R. (2008). Investigating Pair Programming in a Software Engineering Course in an Asian Setting. In: *2008 15th Asia-Pacific Software Engineering Conference*. 2008 15th Asia-Pacific Software Engineering Conference. Beijing, China: IEEE, pp.325–331. Available from: http://ieeexplore.ieee.org/document/4724563/ [accessed 21 May 2022].

Sison, R. (2009). Investigating the Effect of Pair Programming and Software Size on Software Quality and Programmer Productivity. In: *2009 16th Asia-Pacific Software Engineering Conference*. 2009 16th Asia-Pacific Software Engineering Conference (APSEC). Batu Ferringhi, Penang, Malaysia: IEEE, pp.187 – 193. Available from: http://ieeexplore.ieee.org/document/5358611/ [accessed 19 May 2022].

Slyngstad, O.P.N., Li, J., Conradi, R., Rønneberg, H., Landre, E. and Wesenberg, H. (2008). The Impact of Test Driven Development on the Evolution of a Reusable Framework of Components – An Industrial Case Study. In: *2008 The Third International Conference on Software Engineering Advances*. 2008 The Third International Conference on Software Engineering Advances. pp.214–223.

Smart, F.J. (2014). *BDD in Action - Behaviour-Driven Development for the whole software lifecycle*. Manning Publications. Available from: https://manning-content.s3.amazonaws.com/download/a/bdba704-bf94-4c86-ac9a-a8c9353bee14/BDD_CH01.pdf.

Smith, S. and Stoecklin, S. (2001). What we can learn from extreme programming. *J. Comput. Sci. Coll.*, 17(2), pp.144–151.

Snyder, H. (2019). Literature review as a research methodology: An overview and guidelines. *Journal of Business Research*, 104, pp.333–339.

Sobral, S.R. (2020). Is Pair Programming in Higher Education a Good Strategy? *International Journal of Information and Education Technology*, 10(12), pp.911–916.

Software as a Medical Device Working Group. (2017). *Software as a Medical Device (SaMD): Clinical Evaluation*. International Medical Device Regulatory Forum.

Sonigara, B., Sezer, S., Siddiqui, F., Weber, R., Antonopoulos, K., Panagiotou, C., Antonopoulos, C.P., Keramidas, G., Voros, N., Yengec-Tasdemir, S.B., Hui, H. and McLaughlin, K. (2023). XANDAR: Verification & Validation Approach for Safety-critical Systems. In: *2023 IEEE 36th International System-on-Chip Conference (SOCC)*. 2023 IEEE 36th International System-on-Chip Conference (SOCC). Santa Clara, CA, USA: IEEE, pp.1–6. Available from: https://ieeexplore.ieee.org/document/10257177/ [accessed 8 October 2025].

Souza, R., Chavez, C. and Bittencourt, R.A. (2014). Do Rapid Releases Affect Bug Reopening? A Case Study of Firefox. In: *2014 Brazilian Symposium on Software Engineering*. 2014 Brazilian Symposium on Software Engineering (SBES). Maceio, Brazil: IEEE, pp.31–40. Available from: http://ieeexplore.ieee.org/document/6943480/ [accessed 20 April 2023].

Sowri Raja Pillai, N. and Rani Hemamalini, R. (2022). Hybrid User Acceptance Test Procedure to Improve the Software Quality. *The International Arab Journal of Information Technology* [online], 19(6). Available from: http://iajit.org/portal/images/Year2022/No.6/20887.pdf [accessed 3 April 2023].

Spadini, D., Aniche, M., Storey, M.-A., Bruntink, M. and Bacchelli, A. (2018). When Testing Meets Code Review: Why and How Developers Review Tests. In: *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE). pp.677–687.

Spence, J.W. (2005). There Has to Be a Better Way! 2005, p.7.

Squire, C.M., Giombi, K.C., Rupert, D.J., Amoozegar, J. and Williams, P. (2024). Determining an Appropriate Sample Size for Qualitative Interviews to Achieve True and Near Code Saturation: Secondary Analysis of Data. *Journal of Medical Internet Research*, 26, p.e52998.

Sripada, S., Reddy, Y.R. and Sureka, A. (2015). In Support of Peer Code Review and Inspection in an Undergraduate Software Engineering Course. In: *2015 IEEE 28th Conference on Software Engineering Education and Training*. 2015 IEEE 28th Conference on Software Engineering Education and Training (CSEET). Florence: IEEE, pp.3–6. Available from: https://ieeexplore.ieee.org/document/7203843/ [accessed 24 August 2022].

Stapel, K., Knauss, E., Schneider, K. and Becker, M. (2010). Towards Understanding Communication Structure in Pair Programming. In: Sillitti, A. et al., eds *Agile Processes in Software Engineering and Extreme Programming*. Lecture Notes in Business Information Processing. Berlin, Heidelberg: Springer, pp.117–131.

Staron, M., Meding, W. and Söderqvist, B. (2010). A method for forecasting defect backlog in large streamline software development projects and its industrial evaluation. *Information and Software Technology*, 52(10), pp.1069–1079.

Stenfors, T., Kajamaa, A. and Bennett, D. (2020). How to … assess the quality of qualitative research. *The Clinical Teacher*, 17(6), pp.596–599.

Stratton, S.J. (2023). Population Sampling: Probability and Non-Probability Techniques. *Prehospital and Disaster Medicine*, 38(2), pp.147–148.

Stray, V., Moe, N.B. and Bergersen, G.R. (2017). Are Daily Stand-up Meetings Valuable? A Survey of Developers in Software Teams. In: Baumeister, H., Lichter, H., and Riebisch, M., eds *Agile Processes in Software Engineering and Extreme Programming*. Lecture Notes in Business Information Processing. Cham: Springer International Publishing, pp.274–281.

Stray, V., Sjøberg, D.I.K. and Dybå, T. (2016). The daily stand-up meeting: A grounded theory study. *Journal of Systems and Software*, 114, pp.101–124.

Stray, V.G., Moe, N.B. and Aurum, A. (2012). Investigating Daily Team Meetings in Agile Software Projects. In: *2012 38th Euromicro Conference on Software Engineering and Advanced Applications*. 2012 38th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA). Cesme, Izmir, Turkey: IEEE, pp.274–281. Available from: http://ieeexplore.ieee.org/document/6328161/ [accessed 14 February 2023].

Suma, V. and Nair, T.R. (2010). Effectiveness Of Defect Prevention in I.T. For Product Development., 21 January 2010. Available from: https://arxiv.org/abs/1001.3725.

Suryantara, I.G.N. and Andry, J.F. (2018). Development of Medical Record with Extreme Programming SDLC. *International Journal of New Media Technology*, 5(1), pp.47–53.

Sutherland, J. and Sutherland, J.J. (2014). *Scrum: the art of doing twice the work in half the time*. First edition. New York: Currency.

Swanlund, S., Wilson, V., Kapoor, V., Gabbard, M. and Srinath, C. (2024). Software Development Life Cycle. *Journal of the Society for Clinical Data Management* [online], 4(1). Available from: https://www.jscdm.org/article/id/343/ [accessed 23 December 2024].

Taherdoost, H. (2016). Sampling Methods in Research Methodology; How to Choose a Sampling Technique for Research. *International Journal of Academic Research in Management*, 5, pp.18–27.

Tang, T., Lim, M.E., Mansfield, E., McLachlan, A. and Quan, S.D. (2018). Clinician user involvement in the real world: Designing an electronic tool to improve interprofessional communication and collaboration in a hospital setting. *International Journal of Medical Informatics*, 110, pp.90–97.

TanLi, M., Zhang, Y., Wang, Y. and Jiang, Y. (2021). Grey-Box Technique of Software Integration Testing Based on Message. *Journal of Physics: Conference Series*, 2025(1), p.012096.

Tanner, K. (2018). Survey designs. In: *Research Methods*. Elsevier, pp.159–192. Available from: https://linkinghub.elsevier.com/retrieve/pii/B9780081022207000066 [accessed 2 November 2024].

Tanwar, D. (2016). Software Development Models and Their Real Aspects., 5(3).

Tatineni, S. (2021). A Comprehensive Overview of DevOps and Its Operational Strategies. *A Comprehensive Overview of DevOps and Its Operational Strategies. International Journal of Information Technology and Management Information Systems*, 12(1), pp.15–32.

Tavares, H.L., Rezende, G.G., Manhães, R.S. and de Carvalho, R.A. (2010). A tool stack for implementing Behaviour-Driven Development in Python Language., 2010.

Therapeutic Goods Administration (TGA). (2020). Actual and potential harm caused by medical software: A rapid literature review of safety and performance issues., 2020, p.15.

Thompson, C. and Wagner, D. (2017). A Large-Scale Study of Modern Code Review and Security in Open Source Projects. In: *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering*. PROMISE: Predictive Modelling in Software Engineering. Toronto Canada: ACM, pp.83–92. Available from: https://dl.acm.org/doi/10.1145/3127005.3127014 [accessed 17 August 2022].

Thompson, H. (2018). *Software is a top cause of medical device recalls: Here's what you can do* [online]. *Medical Design and Outsourcing* [online]. Available from: https://www.medicaldesignandoutsourcing.com/software-leading-cause-medical-device-recalls/ [accessed 19 February 2022].

Thongtanunam, P., McIntosh, S., Hassan, A.E. and Iida, H. (2015). Investigating Code Review Practices in Defective Files: An Empirical Study of the Qt System. In: *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories (MSR). Florence, Italy: IEEE, pp.168–179. Available from: http://ieeexplore.ieee.org/document/7180077/ [accessed 26 August 2022].

Thongtanunam, P., McIntosh, S., Hassan, A.E. and Iida, H. (2016). Revisiting code ownership and its relationship with software quality in the scope of modern code review. In: *Proceedings of the 38th International Conference on Software Engineering*. ICSE '16: 38th International Conference on Software Engineering. Austin Texas: ACM, pp.1039–1050. Available from: https://dl.acm.org/doi/10.1145/2884781.2884852 [accessed 2 September 2022].

Tianual, P. and Pohthong, A. (2019). Defects Detection Technique of Use Case Views during Requirements Engineering. In: *Proceedings of the 2019 8th International Conference on Software and Computer Applications*. ICSCA '19: 2019 8th International Conference on Software and Computer Applications. Penang Malaysia: ACM, pp.277–281. Available from: https://dl.acm.org/doi/10.1145/3316615.3316631 [accessed 23 March 2023].

Tomayko, J.E. (2002). A Comparison of Pair Programming to Inspections for Software Defect Reduction. *Computer Science Education*, 12(3), pp.213–222.

Tongco, Ma.D.C. (2007). Purposive Sampling as a Tool for Informant Selection. *Ethnobotany Research and Applications*, 5, p.147.

Törner, F., Ivarsson, M., Pettersson, F. and Öhman, P. (2006). Defects in Automotive Use Cases., 2006. Available from: https://www.google.com/url?sa=i&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=0CAMQw7AJahcKEwjoq7Tbi_L9AhUAAAAAHQAAAAQAw&url=https%3A%2F%2Fdl.acm.org%2Fdoi%2Fpdf%2F10.1145%2F1159733.1159753&psig=AOvVaw1zLpDAwJs6iWoIEnpmkjdg&ust=1679662273248134.

Tosun, A., Dieste, O., Fucci, D., Vegas, S., Turhan, B., Erdogmus, H., Santos, A., Oivo, M., Toro, K., Jarvinen, J. and Juristo, N. (2017). An industry experiment on the effects of test-driven development on external quality and productivity. *Empirical Software Engineering*, 22(6), pp.2763–2805.

Tosun, A., Ahmed, M., Turhan, B. and Juristo, N. (2018). On the effectiveness of unit tests in test-driven development. In: *Proceedings of the 2018 International Conference on Software and System Process*. ICSSP '18. New York, NY, USA: Association for Computing Machinery, pp.113–122. Available from: https://doi.org/10.1145/3202710.3203153 [accessed 12 June 2022].

Trautsch, F., Herbold, S. and Grabowski, J. (2020). Are unit and integration test definitions still valid for modern Java projects? An empirical study on open-source projects. *Journal of Systems and Software*, 159, p.110421.

Trivedi, D. (2021). Agile Methodologies. *International Journal of Computer Science & Communication*, 12, pp.91–100.

Turk, D., France. Robert and Rumpe, B. (2005). Assumptions Underlying Agile Software-Development Processes. *Journal of Database Management*, 16(4), pp.62–87.

Uludağ, Ö., Putta, A., Paasivaara, M. and Matthes, F. (2021). Evolution of the Agile Scaling Frameworks. In: Gregory, P. et al., eds *Agile Processes in Software Engineering and Extreme Programming*. Lecture Notes in Business Information Processing. Cham: Springer International Publishing, pp.123–139. Available from: https://link.springer.com/10.1007/978-3-030-78098-2_8 [accessed 22 April 2025].

Vanhanen, J. and Korpi, H. (2007). Experiences of Using Pair Programming in an Agile Project. In: *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*. 2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07). pp.274b–274b.

Vanhanen, J. and Lassenius, C. (2005). Effects of pair programming at the development team level: an experiment. In: *2005 International Symposium on Empirical Software Engineering, 2005.* 2005 International Symposium on Empirical Software Engineering, 2005. p.10 pp.-.

Vanhanen, J. and Lassenius, C. (2007). Perceived Effects of Pair Programming in an Industrial Context. In: *33rd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO 2007)*. 33rd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO 2007). Lubeck, Germany: IEEE, pp.211–218. Available from: http://ieeexplore.ieee.org/document/4301082/ [accessed 11 May 2022].

Vanhanen, J. and Mäntylä, M.V. (2013). A systematic mapping study of empirical studies on the use of pair programming in industry. *International Journal of Software Engineering and Knowledge Engineering*, 23(09), pp.1221–1267.

Vasilescu, B., van Schuylenburg, S., Wulms, J., Serebrenik, A. and van den Brand, M.G.J. (2014). Continuous Integration in a Social-Coding World: Empirical Evidence from GitHub. In: *2014 IEEE International Conference on Software Maintenance and Evolution*. 2014 IEEE 30th International Conference on Software Maintenance and Evolution (ICSME). Victoria, BC: IEEE, pp.401–405. Available from: https://ieeexplore.ieee.org/document/6976106/ [accessed 21 September 2022].

Vasilescu, B., Yu, Y., Wang, H., Devanbu, P. and Filkov, V. (2015). Quality and productivity outcomes relating to continuous integration in GitHub. In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ESEC/FSE'15: Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering. Bergamo Italy: ACM, pp.805–816. Available from: https://dl.acm.org/doi/10.1145/2786805.2786850 [accessed 22 September 2022].

Vassallo, C. (2019). Enabling Continuous Improvement of a Continuous Integration Process. In: *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). San Diego, CA, USA: IEEE, pp.1246 – 1249. Available from: https://ieeexplore.ieee.org/document/8952505/ [accessed 22 September 2022].

Ventresca, M.J. and Mohr, J.W. (2017). Archival Research Methods. In: Baum, J. A. C., ed. *The Blackwell Companion to Organizations*. Oxford, UK: Blackwell Publishing Ltd, pp.805–828. Available from: https://onlinelibrary.wiley.com/doi/10.1002/9781405164061.ch35 [accessed 18 November 2022].

Vijay, D. and Ganapathy, G. (2014). Guidelines To Minimize the Cost of Software Quality in Agile Scrum Process. *International Journal of Software Engineering*, 2014.

Vonken, F. and Zaidman, A. (2012). Refactoring with Unit Testing: A Match Made in Heaven? In: *2012 19th Working Conference on Reverse Engineering*. 2012 19th Working Conference on Reverse Engineering (WCRE). Kingston, ON, Canada: IEEE, pp.29–38. Available from: http://ieeexplore.ieee.org/document/6385099/ [accessed 20 July 2022].

W3C. (2008). Web Content Accessibility Guidelines (WCAG) 2.0., 11 December 2008. Available from: https://www.w3.org/.

Wagner, S. (2008). Defect classification and defect types revisited. In: *Proceedings of the 2008 workshop on Defects in large software systems - DEFECTS '08*. the 2008 workshop. Seattle, Washington: ACM Press, p.39. Available from: http://portal.acm.org/citation.cfm?doid=1390817.1390829 [accessed 21 February 2022].

Wake, W.C. (2002). *Extreme Programming Explored*. Addison-Wesley.

Wallace, D.R. and Kuhn, D.R. (2001). Failure Modes in Medical Device Software: An Analysis of 15 Years of Recall Data. *International Journal of Reliability, Quality and Safety Engineering*, 08(04), pp.351–371.

Wan, Z., Zhang, Y., Xia, X., Jiang, Y. and Lo, D. (2023). Software Architecture in Practice: Challenges and Opportunities. In: *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE '23: 31st ACM Joint European Software Engineering Conference and Symposium

on the Foundations of Software Engineering. San Francisco CA USA: ACM, pp.1457–1469. Available from: https://dl.acm.org/doi/10.1145/3611643.3616367 [accessed 18 March 2024].

Wang, L., Li, B., Hu, B., Shen, G., Zheng, Yunxin and Zheng, Yuanyi. (2022). Failure mode effect and criticality analysis of ultrasound device by classification tracking. *BMC Health Services Research*, 22(1), p.429.

Wang, Y. and Wagner, S. (2018). Combining STPA and BDD for Safety Analysis and Verification in Agile Development: A Controlled Experiment. , 5 April 2018. Available from: http://arxiv.org/abs/1804.01715 [accessed 1 February 2023].

Wang, Y., Zheng, B. and Huang, H. (2008). Complying with Coding Standards or Retaining Programming Style: A Quality Outlook at Source Code Level. *Journal of Software Engineering and Applications*, 01(01), pp.88–91.

Webster, J. and Watson, R.T. (2002). Analyzing the Past to Prepare for the Future: Writing a Literature Review. *MIS Quarterly*, 26(2), pp.xiii–xxiii.

Wedawatta, G., Ingirige, B. and Amaratunga, D. (2011). Case study as a research strategy: Investigating extreme weather resilience of construction SMEs in the UK., 2011, p.10.

Weißgerber, P. and Diehl, S. (2006). Are refactorings less error-prone than other changes? In: *Proceedings of the 2006 international workshop on Mining software repositories - MSR '06*. the 2006 international workshop. Shanghai, China: ACM Press, p.112. Available from: http://portal.acm.org/citation.cfm?doid=1137983.1138011 [accessed 26 July 2022].

Weißgerber, P., Diehl, S. and Görg, C. (2006). Mining refactorings in ARGOUML. In: *Proceedings of the 2006 international workshop on Mining software repositories - MSR '06*. the 2006 international workshop. Shanghai, China: ACM Press, p.175. Available from: http://portal.acm.org/citation.cfm?doid=1137983.1138028 [accessed 27 July 2022].

Weyrauch, K. (2006). What Are We Arguing About? A Framework for Defining Agile in our Organization. In: *AGILE 2006 (AGILE'06)*. Minneapolis, MN, USA: IEEE, pp.213–220. Available from: http://ieeexplore.ieee.org/document/1667582/ [accessed 6 December 2021].

Wiesche, M. (2021). Interruptions in Agile Software Development Teams. *Project Management Journal*, 52(2), pp.210–222.

Wilkerson, J.W., Nunamaker, J.F. and Mercer, R. (2012). Comparing the Defect Reduction Benefits of Code Inspection and Test-Driven Development. *IEEE Transactions on Software Engineering*, 38(3), pp.547–560.

Wilking, D., Khan, U.F. and Kowalewski, S. (2007). An Empirical Evaluation of Refactoring., 2007, p.16.

Williams, L. (2001). Integrating pair programming into a software development process. In: *Proceedings 14th Conference on Software Engineering Education and Training. 'In search of a software engineering profession' (Cat. No.PR01059)*. 14th Conference on Software Engineering Education and Training 'In search of a software engineering profession'. Charlotte, NC, USA: IEEE Comput. Soc, pp.27–36. Available from: http://ieeexplore.ieee.org/document/913816/ [accessed 24 October 2022].

Williams, L., Kessler, R.R., Cunningham, W. and Jeffries, R. (2000). Strengthening the case for pair programming. *IEEE Software*, 17(4), pp.19–25.

Williams, L., Kudrjavets, G. and Nagappan, N. (2009). On the Effectiveness of Unit Test Automation at Microsoft. In: *2009 20th International Symposium on Software Reliability Engi neering*. 2009 20th International Symposium on Software Reliability Engineering. pp.81–89.

Williams, L., Maximilien, E.M. and Vouk, M. (2003). Test-driven development as a defect-reduction practice. In: *14th International Symposium on Software Reliability Engineering, 2003. ISSRE 2003.* 14th International Symposium on Software Reliability Engineering, 2003. ISSRE 2003. Denver, Colorado, USA: IEEE, pp.34–45. Available from: http://ieeexplore.ieee.org/document/1251029/ [accessed 2 May 2022].

Williams, L.A. and Kessler, R.R. (2001). Experimenting with Industry's "Pair-Programming" Model in the Computer Science Classroom., 2001, p.11.

Winkler, D., Schmidt, M., Ramler, R. and Biffl, S. (2012). Improving Unfamiliar Code with Unit Tests: An Empirical Investigation on Tool-Supported and Human-Based Testing. In: Dieste, O., Jedlitschka, A., and Juristo, N., eds *Product-Focused Software Process Improvement*. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, pp.290–304. Available from: http://link.springer.com/10.1007/978-3-642-31063-8_22 [accessed 21 July 2022].

Winkler, D., Kitzler, M., Steindl, C. and Biffl, S. (2013). Investigating the Impact of Experience and Solo/Pair Programming on Coding Efficiency: Results and Experiences from Coding Contests. In: Baumeister, H. and Weber, B., eds *Agile Processes in Software Engineering and Extreme Programming*. Lecture Notes in Business Information Processing. Berlin, Heidelberg: Springer Berlin Heidelberg, pp.106–120. Available from: http://link.springer.com/10.1007/978-3-642-38314-4_8 [accessed 25 October 2022].

Winkler, D. and Biffl, S. (2006). An Empirical Study on Design Quality Improvement from Best-Practice Inspection and Pair Programming. In: Münch, J. and Vierimaa, M., eds *Product-Focused Software Process Improvement*. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, pp.319–333. Available from: http://link.springer.com/10.1007/11767718_27 [accessed 21 May 2022].

Wong, W.E., Li, X. and Laplante, P.A. (2017). Be more familiar with our enemies and pave the way forward: A review of the roles bugs played in software failures. *Journal of Systems and Software*, 133, pp.68–94.

Writing Commons. (2022). *Postpositivism* [online]. *Writing Commons* [online]. Available from: https://writingcommons.org/section/writing-studies-definition/epistemology/postpositivists/ [accessed 10 November 2022].

Xu, S. and Li, T. (2009). Evaluation of Test-Driven Development: An Academic Case Study. In: Lee, R. and Ishii, N., eds *Software Engineering Research, Management and Applications 2009*. Studies in Computational Intelligence. Berlin, Heidelberg: Springer, pp.229–238. Available from: https://doi.org/10.1007/978-3-642-05441-9_20 [accessed 9 June 2022].

Yadav, A.R., Pawar, A.R., Shete, A.S., Jadhav, N.R., Kumbhar, A.R., Jadhav, N.M. and Gejage, S.M. (2025). Comparative analysis of medical device adverse event reporting forms

for patients and industries across various countries, and the introduction of a unified generic form for harmonization. *Expert Review of Medical Devices*, 22(1), pp.49–62.

Yadav, K.S. and Yasvi, M.A. (2019). Review On Extreme Programming-XP. *International Conference on Robotics, Smart Technology and Electronics Engineering*, 2019.

Yaman, S.G., Sauvola, T., Riungu-Kalliosaari, L., Hokkanen, L., Kuvaja, P., Oivo, M. and Männistö, T. (2016). Customer Involvement in Continuous Deployment: A Systematic Literature Review. In: Daneva, M. and Pastor, O., eds *Requirements Engineering: Foundation for Software Quality*. Lecture Notes in Computer Science. Cham: Springer International Publishing, pp.249–265. Available from: http://link.springer.com/10.1007/978-3-319-30282-9_18 [accessed 30 September 2022].

Yarlagadda, R.T. (2017). Implementation of DevOps in healthcare systems., 4(6), p.6.

Yin, X., Knight, J. and Weimer, W. (2009). Exploiting refactoring in formal verification. In: *2009 IEEE/IFIP International Conference on Dependable Systems & Networks*. Networks (DSN). Lisbon, Portugal: IEEE, pp.53 – 62. Available from: http://ieeexplore.ieee.org/document/5270355/ [accessed 6 August 2022].

Ynchausti, R. (2001). Integrating Unit Testing into a Software Development Team's Process., 29 August 2001.

Yogesh, T. and Vimala, P. (2020). Test-Driven Development of Automotive Software Functionality. In: *2020 Third International Conference on Smart Systems and Inventive Technology (ICSSIT)*. 2020 Third International Conference on Smart Systems and Inventive Technology (ICSSIT). pp.1162–1165.

Yoon, C., Nam, K.C., Lee, Y.K., Kang, Y., Choi, S.J., Shin, H.M., Jang, H., Kim, J.K., Kwon, B.S., Ishikawa, H. and Woo, E. (2019). Differences in Perspectives of Medical Device Adverse Events: Observational Results in Training Program Using Virtual Cases. *Journal of Korean Medical Science*, 34(39), p.e255.

Yu, J. (2018). Research Process on Software Development Model. *IOP Conference Series: Materials Science and Engineering*, 394, p.032045.

Yuksel, H.M., Tuzun, E., Gelirli, E., Biyikli, E. and Baykal, B. (2009). Using continuous integration and automated test techniques for a robust C4ISR system. In: *2009 24th International Symposium on Computer and Information Sciences*. 2009 24th International Symposium on Computer and Information Sciences. pp.743–748.

Yusoff, M.S.B. (2019). ABC of Content Validation and Content Validity Index Calculation. *Education in Medicine Journal*, 11(2), pp.49–54.

Zaidman, M. (2004). Teaching defensive programming in Java. *Journal of Computing Sciences in Colleges*, 19(3), pp.33–43.

Zalaghi, H. and Khazaei, M. (2016). The Role of Deductive and Inductive Reasoning in Accounting Research and Standard Setting. *Asian Journal of Finance & Accounting*, 8(1), p.23.

Zamith, M. and Gonçalves, G. (2018). Towards an Agile Development Model for Certifiable Medical Device Software - Taking Advantage of the Medical Device Regulation. In: *ICSOFT*.

Zhang, Y., Masci, P., Jones, P. and Thimbleby, H. (2019). User Interface Software Errors in Medical Devices: Study of U.S. Recall Data. *Biomedical Instrumentation & Technology*, 53(3), pp.182–194.

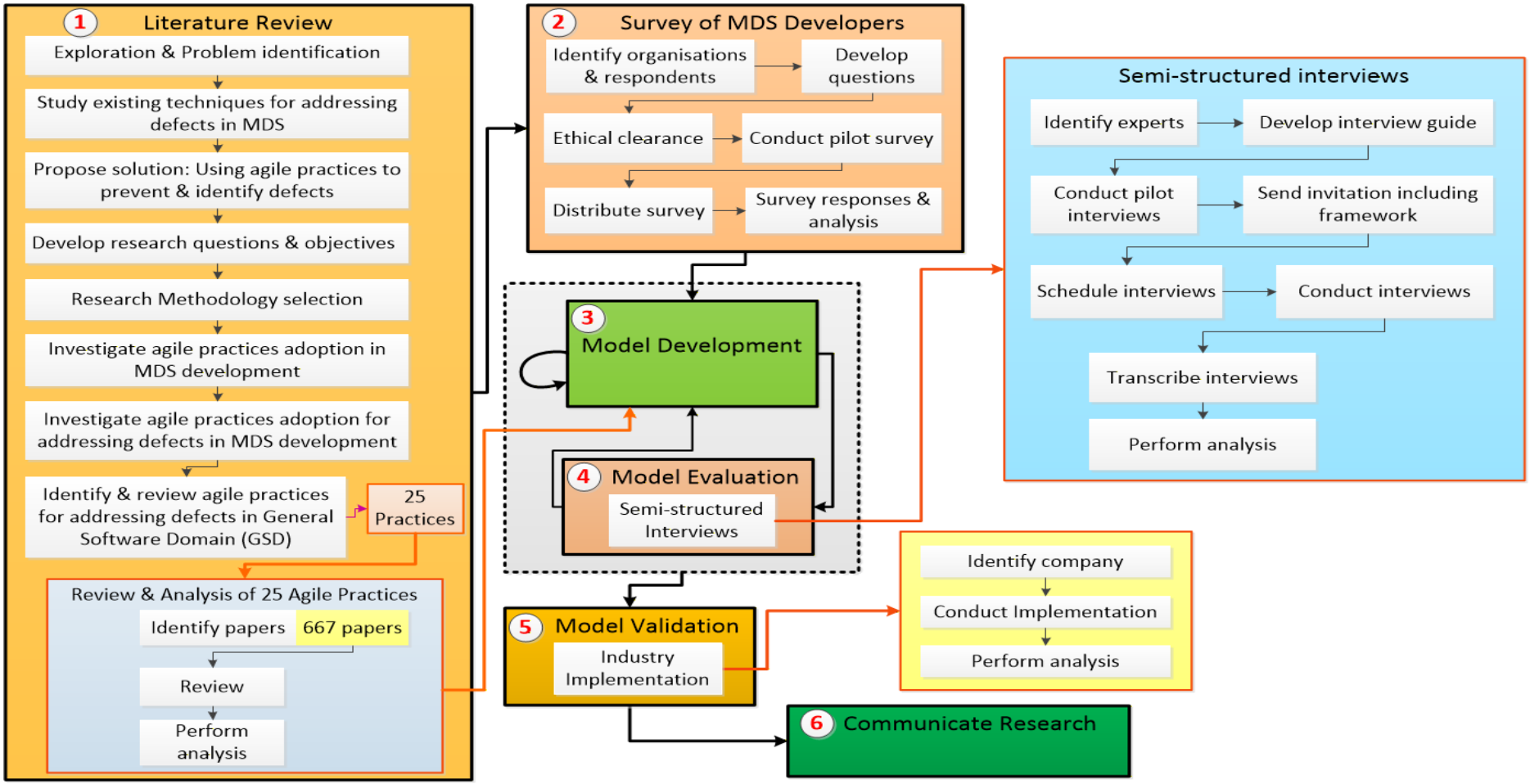
Zhao, Y., Serebrenik, A., Zhou, Y., Filkov, V. and Vasilescu, B. (2017). The impact of continuous integration on other software development practices: A large-scale empirical study. In: *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE). Urbana, IL: IEEE, pp.60 – 71. Available from: http://ieeexplore.ieee.org/document/8115619/ [accessed 26 September 2022].

Zohrabi, M. (2013). Mixed Method Research: Instruments, Validity, Reliability and Reporting Findings. *Theory and Practice in Language Studies*, 3(2), pp.254–262.

Žukauskas, P., Vveinhardt, J. and Andriukaitienė, R. (2018). Philosophy and Paradigm of Scientific Research. In: Žukauskas, P., Vveinhardt, J., and Andriukaitien, R., eds *Management Culture and Corporate Social Responsibility*. InTech. Available from: http://www.intechopen.com/books/management-culture-and-corporate-social-responsibility/philosophy-and-paradigm-of-scientific-research [accessed 9 November 2022].

8 Appendices

Appendix A: Research process Steps



Appendix B: Studies from the literature review and Analysis

B.1 – Studies reporting improvements in defects for given practice

Agile practice	Papers
Test-Driven Development	(Lui and Chan 2004; Damm et al. 2005; George and Williams 2004; Bhat and Nagappan 2006; Papis et al. 2022; Latorre 2014; Crispin 2006; Sanchez et al. 2008; Maximilien and Williams 2003; Braga et al. 2015; Čaušević et al. 2013; Čaušević et al. 2012; Cauevic et al. 2012; Tosun et al. 2018; Bhadauria et al. 2020; Slyngstad et al. 2008; Karlesky et al. 2006; Edwards 2003; Moe and Oo 2020; Marchenko et al. 2009; Buchan et al. 2011; Sanchez et al. 2007; Williams et al. 2003; Bakhtiary et al. 2020; Amrit and Meijberg 2018; Dogša and Batič 2011; Nagappan et al. 2008; Rayudu and Kumar 2014; Xu and Li 2009; Janzen and Saiedian 2006; Flohr and Schneider 2006; Janzen 2005; Janzen and Saiedian 2007; Scanniello et al. 2016)
Unit Testing	(Williams et al. 2009; Winkler et al. 2012; Ramler et al. 2013; Daka and Fraser 2014; Delgado and Martinez 2013; Runeson and Andrews 2003; Ellims et al. 2004; Ellims et al. 2006; Seshu Kumar and Vasavi 2012; Ynchausti 2001; Bacchelli et al. 2008; Serra et al. 2019; Klammer and Kern 2015; Orellana et al. 2017; Dudila and Letia 2013; Mockus et al. 2009; Winkler et al. 2012; Tosun et al. 2018; Sabane et al. 2013; Scott and Wohlin 2008; Winkler et al. 2012)
Refactoring	(Gatrell and Counsell 2015; Kim et al. 2012; Geppert et al. 2005), (Chug and Gupta 2017; Weißgerber et al. 2006; Ratzinger et al. 2008; Di Penta et al. 2020; Ma et al. 2016, p.20; Palomba et al. 2017; Bavota et al. 2015; Amit and Feitelson 2019; AlOmar et al. 2021), (Kannangara 2013; Wilking et al. 2007; Yin et al. 2009; Ouni et al. 2016; Ouni et al. 2013; Reddy and Rao 2009), (Bagheri and Hegedus 2022), Görg and Weißgerber (2005), Ferreira et al. (2018), Weißgerber and Diehl (2006)
Code Review	(Shimagaki et al. 2016; Q. Fu et al. 2017; Czerwonka et al. 2015; Bacchelli and Bird 2013; Rigby and Bird 2013), (Thompson and Wagner 2017; McIntosh et al. 2014; McIntosh et al. 2016; Morales et al. 2015; Bavota and Russo 2015; Thongtanunam et al. 2015; Thongtanunam et al. 2016; Beller et al. 2014; Pascarella et al. 2019; Paul et al. 2021), (Sripada et al. 2015). Mantyla and Lassenius (2009), Spadini et al. (2018)
Pair Programming	(Vanhanen and Korpi 2007; Phaphoom et al. 2011; Q. Fu et al. 2017; di Bella et al. 2013; Jensen 2003; Vanhanen and Lassenius 2007; Phongpaibul and Boehm 2007; Vanhanen and Lassenius 2005; Sison 2009; Tomayko 2002; Balijepally et al. 2009; Sison 2008; Padmanabhuni et al. 2012; Williams et al. 2000; Phongpaibul and Boehm 2006; Winkler et al. 2013)
Continuous Integration	(Amrit and Meijberg 2018; Yuksel et al. 2009), (Vasilescu et al. 2015; Keskin Kaynak et al. 2019; Mellegard et al. 2020; Zhao et al. 2017; Rahman et al. 2018), (Hilton et al. 2016; Pinto et al. 2018), Vassallo (2019)
Code Ownership	Bird et al. (2010), Bird et al. (2011), Greiler et al. (2015), Faragó et al. (2015), Foucault et al. (2014)
Coding Standards	(Boogerd and Moonen 2008), (Popic et al. 2018), Dorin (2018), Capiluppi et al. (2009), Zaidman (2004)
Small Releases	(Khomh et al. 2012; Souza et al. 2014; Abou Khalil et al. 2021; Abou Khalil et al. 2019)
Behaviour Driven Development	Ali (2019), Irshad et al. (2021)
Onsite Customer	(Tang et al. 2018), Korkala et al. (2006), Kabbedijk et al. (2009)
User Acceptance Testing	Jeeva Padmini et al. (2016), Hugo Santiago C. Pinto et al. (2018), Sowri Raja Pillai and Rani Hemamalini (2022)
Integration Testing	(Orellana et al. 2017), (Holling et al. 2016)
User Story	Scott et al. (2021), Lucassen, Dalpiaz, Werf, et al. (2016)
Simple Design	(D'Ambros et al. 2010)

B.2 – Studies reporting no improvements in defects for given practice

Agile practice	Papers
Test-Driven Development	(Wilkerson et al. 2012), (Munir et al. 2014)
Unit Testing	(Petrić et al. 2020; Gren and Antinyan 2017; Antinyan et al. 2018; Jalote and Haragopal 1998; Vonken and Zaidman 2012).
Refactoring	Bavota et al. (2012)
Code Review	(Kononenko et al. 2015)
Pair Programming	Hulkko and Abrahamsson (2005), Madeyski (2008)
Small Releases	(Jansen et al. 2021)

B.3 – Summary of findings on agile practices

Practice	Study type	Improvement	No improvement	No difference
UAT	Case study	3	0	0
	Total	**3**	**0**	**0**
BDD	Experiment	1	0	1
	Total	**1**	**0**	**1**
Small Releases	Case study	4	0	1
	Total	**4**	**0**	**1**
Coding Standards	Experiment	2	0	0
	Case study	3	0	0
	Total	**5**	**0**	**0**
Onsite Customer	Experience report	1	0	0
	Case study	1	0	0
	Case study & Interview	1	0	0
	Total	**3**	**0**	**0**
Code Ownership	Case study	5	0	0
	Total	**5**	**0**	**0**

Appendix C: Materials for Survey

C.1 Participant Information Leaflet

Study title: Developing an Agile-based Framework to Address Defects in Medical Device Software Development

Researcher Name: Misheck Nyirenda

Telephone number of Researcher: 08922XXXXX

Research Supervisor Name: Dr. Martin Mc Hugh & Dr. Róisín Loughran

You are being invited to take part in a research study to be carried out at Dundalk Institute of Technology (DkIT).

Before you decide whether or not you wish to take part, you should read the information provided below carefully.

You should clearly understand the risks and benefits of taking part in this study so that you can make a decision that is right for you. This process is known as 'Informed Consent'.

You don't have to take part in this study and the decision not to take part will have no effect on your future.

You can change your mind about taking part in the study any time you like. Even if the study has started, you can still opt out. You don't have to give us a reason.

Why is this study being done?

This research is being undertaken to gather information from medical device software developers about the agile practices they currently use to address defects when developing medical device software and how they implement them. The information from this research will be correlated with the literature review findings to develop an agile based framework. This framework will help developers to apply agile practices to address defects when developing medical device software.

Who is organising and funding this study?

This research is being carried out by Misheck Nyirenda, a doctoral researcher in the Regulated Software Research Centre at Dundalk Institute of Technology (DkIT). This research is funded by the HEA landscape, TUTF and DkIT. The research contributes to a PhD in Software Engineering.

Why am I being asked to take part?

You are being asked to participate in this research because of your expertise in medical device software development. We believe that your unique insights and experiences are valuable to this research.

How will the study be carried out?

As a participant in this research, your role involves completing an online survey. The survey will be accessed via a link that will be provided to you, including the timeline indicating the period in which the survey needs to be completed. Clarification on certain responses may be sought from some participants.

What are the benefits?

Your unique insights and experiences are useful and will contribute to the development of a comprehensive agile based framework to address defects when developing medical device software. This will benefit medical device software developers in using agile practices to address defects when developing medical device software.

What are the risks?

This research only requires approximately 15 minutes of your time to complete the survey.

Is the study confidential?

This research is confidential; the survey does not collect information such as name of respondent and their organisation. The information collected in this research will only be accessible to people directly involved in this research.

Where can I get further information?

If you have any further questions about the study or if you want to opt out of the study, you can rest assured it won't affect the quality of treatment you get in the future.

If you need any further information now or at any time in the future, please contact:

Name: Misheck Nyirenda

Address: Dundalk Institute of Technology, Dublin Road, Dundalk, Co Louth.

Phone No: 429370200

D.2 Participants Consent Form

Study title: Developing an Agile-based Framework to Adress Defects in Medical Device Software Development

I have read and understood the Information Leaflet about this research project. The information has been fully explained to me, and I have been able to ask questions, all of which have been answered to my satisfaction.	Yes □	No □
I understand that I don't have to take part in this study and that I can opt out at any time. I understand that I don't have to give a reason for opting out and I understand that opting out won't affect my future.	Yes □	No □
I am aware of the potential risks of this research study.	Yes □	No □
I have been assured that information about me will be kept private and confidential.	Yes □	No □
I have been given a copy of the Information Leaflet and this completed consent form for my records.	Yes □	No □
Storage and future use of information:	Yes □	No □

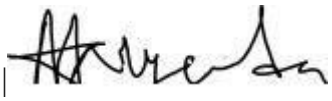
I give my permission for information collected about me to be stored or electronically processed for the purpose of research and to be used in related studies or other studies in the future but only if the research is approved by a Research Ethics Committee.		

| | |

Participant Name (Block Capitals) | Participant Signature | Date

To be completed by the Researcher:

I, the undersigned, have taken the time to fully explain to the above participant the nature and purpose of this study in a way that they could understand. I have explained the risks involved as well as the possible benefits. I have invited them to ask questions on any aspect of the study that concerned them.

MISHECK NYIRENDA | BSc, MSc |  |09/10/2023

Name (Block Capitals) | Qualifications| Signature | Date

C.3 Ethical Approval

Ethical Approval Application - Feedback Form	
Application No.	2.
Date.	13/10/23
Applicants Name.	Misheck Nyirenda
Supervisor.	Dr Martin Mc Hugh & Dr Róisín Loughran
Project Title.	Developing an Agile Based Framework to Address Defects in Medical Device Software Development
Decision. (Approved/ Not Approved/ Approved Subject to indicated Requirements)	Approved Subject to indicated Requirements 1) Correct the End Date of Research in 3.2 (C). Should be 2024. 2) Introductory paragraph at top of questionnaire cuts off abruptly. Complete this.
Comments	

C.4 Survey Questionnaire

Defects in Medical Device Software Development

Thank you for agreeing to participate in our survey. The survey should take you approximately 15 minutes to complete. Please take a moment to read the information in the leaflet. If you would like to be further involved in this research, please complete the consent form and return it to us. We appreciate your time and effort.

* Required

General information

1. How long have you worked in this organization? *

   ○ less than a year

   ○ 1-5 years

   ○ 5-10 years

   ○ 10 -15 years

   ○ More than 15 years

2. What is your current role (position) in this organization? *

   ○ Developer

   ○ Tester

   ○ Developer& Tester

   ○ Project manager

   ○ Other

3. How many years of experience do you have in medical device software development? *

   ○ Less than a year

   ○ 1-5 years

- ○ 5-10 years
- ○ 10-15 years
- ○ More than 15 years

4. How many people work in your organization? *
   - ○ Less than10 people
   - ○ 10-49 people
   - ○ 50-249 people
   - ○ More than 250 people
5. How many people are in the software development team in your organization? *

6. How many years has your organisation been developing medical device software? *
   - ○ 1-5 years
   - ○ 5-10 years
   - ○ 10-15 years
   - ○ 15-20 years
   - ○ More than 20 years

Software development methodology used for medical device software development

7. Which of the following methodologies best describes the approach you are currently following when developing medical device software in your organization? (tick all that apply) *
   - ☐ Agile
   - ☐ Waterfall
   - ☐ V-Model
   - ☐ Other
8. Why do you use this methodology in your organization? (tick all that apply) *

- ☐ To meet regulatory requirements
- ☐ To reduce time to market
- ☐ To reduce development costs
- ☐ To accommodate changing requirements
- ☐ To address defects during development

9. How long has your organization been using this methodology to develop medical device software? *

- ○ Less than 1 year
- ○ 1-5 years
- ○ 5-15 years
- ○ 15-20 years
- ○ More than 20 years

Measures used to address defects in medical device software development

10. How important is it for your organization to prioritize the following aspects when developing medical device software? *

	Extremely important	Very important	Important	Not important	Not important at all
Preventing defects	○	○	○	○	○
Identifying defects	○	○	○	○	○
Preventing defects over identifying defects	○	○	○	○	○

11. At which development phase(s) do you identify defects when developing medical device software in your organization? *

- ☐ Requirements

☐ Design

☐ Coding

☐ Testing

☐ Release

☐ Maintenance

12. Do you use agile practices in your organization to address defects when developing medical device software? *

○ Yes

○ No

13. Which method(s) do you use in your organization to identify defects when developing medical device software? *

☐ Standard test process conducted later in the software development cycle after the product build.

☐ Taxonomy based testing where tests are derived based on defect category using ANSI/AAMI SW91 standard.

☐ Static analysis

☐ Dynamic analysis

☐ Taxonomy based testing where tests are derived based on defect category using generic taxonomies such as Orthogonal Defect Classification & Defect Origins, Types, and Modes.

☐ Other

14. Do you think agile practices can help to address defects at the phase(s) you indicated previously? *

○ Yes

○ No

15. Which agile practices do you think can help to address defects?

16. Please share any comment(s) that you may have about using agile practices to address defects in medical device software development.

Using agile practices to address defects in medical device software development

17. Please list which of the following agile practices you are using to address defects when developing medical device software in your organization. *

- ☐ Pair Programming
- ☐ Refactoring
- ☐ Test-Driven Development
- ☐ Onsite Customer
- ☐ User Acceptance Testing
- ☐ Code Review
- ☐ Continuous Integration
- ☐ Code Ownership
- ☐ Unit Testing
- ☐ Small Releases
- ☐ Coding Standards
- ☐ Integration Testing
- ☐ BDD
- ☐ User Story
- ☐ Use Case
- ☐ Other

18. At which development phase(s) do you use agile practice(s) to address defects in your organization? *

- ☐ Requirements
- ☐ System Design
- ☐ Architectural Design
- ☐ Module Design
- ☐ Coding
- ☐ Testing
- ☐ Release
- ☐ Maintenance

19. Apart from the agile practices you are currently using, which other agile practice(s) do you think can be used to address defects when developing medical device software?

20. Do you have any other comments about using agile practices to address defects in medical device software development? *

- ○ Yes
- ○ No

Please share your comment(s) below. *

We appreciate the time and effort you have dedicated to completing our survey. If you are happy for us to contact you again, please enter your email address below.

Email address

Appendix D: Semi-structured Interview Guide

Interview guide			
A	**Introduction**		
	1. Thank respondents for agreeing to the discussion/interview.		
	2. Brief introduction about myself		
	3. Brief explanation about the project		
	4. Get consent to record and remind respondents that their answers are confidential.		
B	**Overview of AV4AD**		
	(Ask if they would want a presentation)		
	1. Optional PowerPoint presentation to explain AV4AD and terms used		
	i.e., **addressing- prevention** and **identification**		
C	**Exploration of AV4AD**		
	1. Requirements Gathering	**Follow up questions/probes**	**NOTES**
i	What are your thoughts on using **User Story, Onsite Customer,** and **Use Case** to prevent defects in requirements gathering?	May you tell me more about User Story? What about Use Case? May you elaborate on Onsite Customer?	
ii	Can you share your view on using **Onsite Customer/Proxy** to prevent defects in requirements gathering?	Can Onsite Customer/Proxy prevent defects at requirements gathering in your organisation? May you elaborate why you think that way?	
iii	I would like to hear your thoughts on using **Use Case** to prevent defects in requirements gathering.	Can Use Case be useful to prevent defects during requirements gathering in your organisation? May you elaborate why you think so?	
iv	Can you share your opinion on the complementary use of **User Story and Onsite Customer/Proxy** to prevent defects in requirements gathering?	What are your thoughts on the combined use of User Story and Onsite Customer/Proxy to prevent defects during requirements gathering in your organisation? May you elaborate why you feel so?	
v	What is your view on the complementary use of **Use Case and Onsite Customer/Proxy** to prevent defects in requirements gathering?	Can Use Case and Onsite Customer/Proxy be used together to prevent defects during requirements gathering in your organisation? If not, may you explain why you think so.	
vi	Can you share your opinion on the complementary use of **User Story**, **Onsite Customer/Proxy,** and **Use Case** to prevent defects during requirements gathering in your organisation.	In your opinion, can User Story, Onsite Customer/Proxy, and Use Case be used together to prevent defects during requirements gathering in your organisation? If not, may you elaborate why you feel so?	

vii	Have I missed any agile practice that could be useful to prevent defects at this stage?	Which practice? Why do you think it could work at this stage?	
Viii	What improvements or changes would you suggest at this stage?	May you elaborate why you think so	
	2. SW Requirements Analysis		
i	How do you see **Model Storming** as a defect preventing practice in requirements analysis?	Can Model Storming be used to address defects during requirements analysis in your organisation? If not, may you elaborate	
ii	What are your thoughts on using **BDD** to prevent defects in requirements analysis?	Can BDD be used to prevent defects during requirements analysis in your organisation? If not, may explain why you think so?	
iii	Can you share your opinion on using **PP** to prevent defects in requirements analysis?	Can PP be used to prevent defects during requirements analysis in your organisation? If not, may you elaborate why you feel so?	
iv	I would like to hear your thoughts on using **TDD** to prevent defects in requirements analysis.	Can you share your opinion on using TDD to prevent defects during requirements analysis in your organisation? May you elaborate why you feel so?	
v	What is your view on using **Onsite Customer/Proxy** to prevent defects?	Can Onsite Customer be used to prevent defects during requirements analysis in your organisation? May you explain why you think so?	
vi	What are your thoughts on the complementary use of **Model Storming** and **BDD** to prevent defects?	Can Model Storming and BDD be used together to prevent defects during requirements analysis in your organisation? May you explain why you think so?	
vii	What is your opinion on the complementary use of **PP, TDD, and Onsite Customer/Proxy** to prevent defects in requirements analysis?	Can PP, TDD, and Onsite Customer/Proxy be used together to prevent defects during requirements analysis in your organisation? May you explain why you think so?	
Viii	I would like to hear your thoughts on the complementary use of **Model Storming, BDD, PP, TDD, and Onsite Customer** to prevent defects in requirements analysis.	Can Model Storming, BDD, PP, TDD, and Onsite Customer/Proxy be used together to prevent defects during requirements analysis in your organisation. If not, may you elaborate why you feel so?	
ix	Have I missed any agile practice that could be useful to prevent defects at this stage?	which practice? Why do you think it can work at this stage?	

x	What improvements or changes would you suggest at this stage?	May you elaborate why you feel so?	
	3. SW Architectural Design		
i	What is your view on using **Model Storming** to prevent defects in architectural design?	Can Model Storming be used to prevent defects during architectural design in your organisation? If not, may you elaborate	
ii	What are your thoughts on using **Onsite Customer/Proxy** to prevent defects in architectural design?	Can Onsite Customer/Proxy be used to prevent defects during architectural design in your organisation? If not, may you elaborate why you think so?	
iii	What is your opinion on using **Integration Testing** to prevent defects in architectural design?	Can Integration Testing be used to prevent defects during architectural design in your organisation? May you elaborate why you feel so?	
iv	I would like to hear your thoughts on using **Simple Design** to prevent defects in architectural design.	Can Simple Design be used to prevent defects during architectural design in your organisation? May you elaborate why you feel so?	
v	What is your opinion on the complementary use of **Model Storming, Onsite Customer/Proxy, and Integration Testing** to prevent defects in architectural design?	What are your thoughts on using Model Storming, Onsite Customer/Proxy, and Integration Testing together to prevent defects during architectural design in your organisation? May you elaborate why you feel so?	
vi	What are your thoughts on the complementary use of **Simple Design, Onsite Customer/Proxy, and Integration Testing** to prevent defects in architectural design?	Can Simple Design, Onsite Customer/Proxy, and Integration Testing be used together to prevent defects during architectural design in your organisation? May you elaborate why you feel so?	
vii	What is your view on the complementary use of **Model Storming, Onsite Customer/Proxy, Simple Design, and Integration Testing** to prevent defects in architectural design?	Can Model Storming, Onsite Customer/Proxy, Simple Design, and Integration Testing be used to prevent defects during architectural design in your organisation? May you elaborate why you feel so?	
viii	Have I missed any agile practice that could be useful to prevent defects at this stage?	Which practice? Why do you think it could work at this stage?	

ix	What improvements or changes would you suggest at this stage?	May you elaborate why you feel so?	
	4. SW Detailed Design		
i	What are your thoughts on using **Model Storming** to prevent defects in detailed design?	Can Model Storming be used to prevent defects during detailed design in your organisation? If not, may you elaborate?	
ii	What is your view on using **Onsite Customer/Proxy** to prevent defects in detailed design?	Can Onsite Customer/Proxy be used to prevent defects during detailed design in your organisation? If not, may you elaborate?	
iii	What is your opinion on using **Integration Testing** to prevent defects in detailed design?	Can Integration Testing be used to prevent defects during detailed design in your organisation? If not, may you elaborate?	
iv	I would like to hear your thoughts on using **Simple Design** to prevent defects in detailed design.	Can Simple Design be used to prevent defects during detailed design in your organisation? May you elaborate why you feel so?	
V	What are your thoughts on the complementary use of **Model Storming, Onsite Customer/Proxy, and Integration Testing** to prevent defects in detailed design?	Can Model Storming, Onsite Customer/Proxy, and Integration Testing be used together to prevent defects during detailed design in your organisation? May you explain why you feel so?	
Vi	What are your thoughts on the complementary application of **Simple Design, Onsite Customer, and Integration Testing** to prevent defects in detailed design?	Can Simple Design, Onsite Customer, and Integration Testing be used together to prevent defects during detailed design in your organisation? May you elaborate why you feel so?	
vii	I would like to hear your opinion on the complementary use of **Model Storming, Onsite Customer/Proxy, Simple Design, and Integration Testing** to prevent defects in detailed design.	Can Model Storming, Onsite Customer/Proxy, Simple Design, and Integration Testing be used together to prevent defects during detailed design in your organisation? May you elaborate why you feel so?	
Viii	Have I missed any agile practice that could be useful to prevent defects at this stage?	Which practice? Why do you think it could work at this stage?	
ix	What improvements or changes would you suggest at this stage?	May you elaborate why you feel so?	
	5. SW Unit Implementation		

i	What are your thoughts on using **PP** to address defects in SW unit implementation?	Can PP be used to prevent defect during SW unit implementation in your organisation? If not, may you elaborate	
ii	What is your view on using **TDD** to prevent defects during SW unit implementation?	Can TDD be used to prevent defects during SW unit implementation in your organisation? If not, may you elaborate	
iii	What is your opinion on using **Model Storming** to prevent defects in SW unit implementation?	Can Model Storming be used to prevent defects during SW unit implementation in your organisation? May you elaborate why you feel so?	
iv	I would like to hear your thoughts on using **Coding Standards** to prevent defects in SW unit implementation.	Can Coding Standards be used to prevent defects during SW unit implementation in your organisation? May you elaborate why you feel so?	
v	What are your thoughts on using **Onsite Customer/Proxy** to prevent defects in SW unit implementation?	Can Onsite Customer be used to prevent defects during SW unit implementation in your organisation? May you explain why you feel so?	
vi	What is your view on using **Code Ownership** to prevent defects in SW unit implementation?	Can Code Ownership be used to prevent defects during SW unit implementation in your organisation? May you elaborate why you feel so?	
vii	I would like to hear your thoughts on using **Unit Testing** to identify defects in SW unit implementation.	Can Unit Testing identify defects during SW unit implementation in your organisation? May you elaborate why you feel so?	
viii	What are your thoughts on using **Refactoring** to address defects in SW unit implementation?	Can Refactoring be used to address defects during SW unit implementation in your organisation? May you explain why you feel so?	
ix	What is your view on using **Small Release** to address defects in SW unit implementation?	Can Small Release be used to address defects during SW unit implementation in your organisation? May you elaborate why you feel so?	
x	What is your opinion on using **Code Review** to identify defects in SW unit implementation?	Can Code Review be used to identify defects during SW unit implementation in your organisation? May you explain why you feel so?	
xi	What are your thoughts on using **CI** to address defects in SW unit implementation?	Can CI identify be used to address defects during SW unit implementation in your organisation? May you elaborate why you feel so?	
xii	What are your thoughts on the complementary use of **Model Storming, PP, TDD, Onsite Customer/Proxy, Coding Standards, and Code Ownership**	Can Model Storming, PP, TDD, Onsite Customer/Proxy, Coding Standards, and Code Ownership be used together to prevent defects during SW unit implementation in	

	to prevent defects in SW unit implementation?	your organisation? May you elaborate why you feel so?	
xiii	What are your thoughts on the complementary application of **PP, Onsite Customer/Proxy, Unit Testing, Small Release, and Refactoring** to address defects in SW unit implementation?	Can PP, Onsite Customer/Proxy, Unit Testing, Small Release, and Refactoring be used together to address defects during SW unit implementation in your organisation? May you elaborate why you feel so?	
Xi V	What is your view on using **Model Storming**, **PP, TDD, Onsite Customer/Proxy, Coding Standards, Code Ownership, Unit Testing, Small Release, Refactoring, Code Review, and CI** in your organisation to address defects in SW unit implementation?	Can Model Storming, PP, TDD, Onsite Customer/Proxy, Coding Standards, Code Ownership, Unit Testing, Small Release and Refactoring, Code Review and CI be used together to address defects during SW unit implementation in your organisation? May you elaborate why you feel so?	
xv	Have I missed any agile practice that could be useful to address defects at this stage?	Which practice? Why do you think it could work at this stage?	
xvi	What improvements or changes would you suggest at this stage?	May you elaborate why you feel so?	
	6. SW Unit Testing		
i	What are your thoughts on using **Unit Testing** to identify defects in SW unit testing stage?	Can Unit Testing be used to identify defects during SWE unit testing stage in your organisation? If not, may you elaborate	
ii	What is your view on using **Onsite Customer/Proxy** to identify defects in SW unit testing stage?	Can Onsite Customer/Proxy be used to identify defects during SW unit testing in your organisation? May you explain why you feel so?	
iii	What is your opinion on using **PP** to identify defects in SW unit testing?	Can PP be useful to address defects during SW unit testing in your organisation? May you elaborate why you feel so?	
iv	I would like to hear your thoughts on using **Refactoring** to address defects in SW unit testing.	Can Refactoring be used to address defects during SW unit testing in your organisation? May you elaborate why you feel so?	
v	What are your thoughts on the complementary use of **Unit Testing, Onsite Customer/Proxy, PP, and Refactoring** to address defects in SW unit testing?	Can Unit Testing, Onsite Customer/Proxy, PP, and Refactoring be used together to address defects during SW unit testing in your organisation? May you elaborate why you feel so?	

vi	Have I missed any agile practice that could be useful to address defects at this stage?	Which practice? Why do you think it could work at this stage?	
vii	What improvements or changes would you suggest at this stage in relation to addressing defects?	May you elaborate why you feel so?	
	7. SW Integration and Integration Testing		
i	What are your thoughts on using **Integration Testing** to identify defects in SW integration and integration testing stage?	Can Integration Testing be used to identify defects during SW integration and integration testing stage in your organisation? If not, may you elaborate	
ii	What is your view on using **PP** to address defects in SW integration and integration testing stage?	Can PP be used to address defects during SW integration and integration testing stage in your organisation? If not, explain why you think so.	
iii	What is your opinion on using **Refactoring** to identify defects in SW integration and integration testing stage?	Can Refactoring be used to address defects during SW integration and integration testing stage in your organisation? May you elaborate why you feel so?	
iv	I would like to hear your thoughts on using **Unit Testing** to identify defects in SW integration and integration testing stage.	Can Unit Testing be used to identify defects during SW integration and integration testing stage in your organisation? May you elaborate why you feel so?	
v	What is your view on using **Onsite Customer/Proxy** to identify defects in SW integration and integration testing stage?	Can Onsite Customer/Proxy be used to identify defects during SW integration and integration testing stage in your organisation? May you elaborate why you feel so?	
vi	I would like to hear your thoughts on using **UAT** to identify defects in SW integration and integration testing stage.	Can UAT be used to identify defects during SW integration and integration testing stage in your organisation? May you explain why you feel so?	
vii	What are your thoughts on using **CI** to identify defects in SW integration and integration testing stage?	Can CI be used to address defects during SW integration and integration testing stage in your organisation? May you elaborate why you feel so?	
viii	What are your thoughts on the complementary use of **Integration Testing, PP, Refactoring, Unit Testing and CI** to identify defects	Can Integration Testing, PP, Refactoring, Unit Testing and CI be used together to address defects during SW integration and	

	in SW integration and integration testing stage?	integration testing stage in your organisation? May you explain why you feel so?	
ix	What are your thoughts on the complementary application of **UAT and Onsite Customer/Proxy** to identify defects in SW integration and integration testing stage?	Can UAT and Onsite Customer/Proxy be used together to identify defects during SW integration and integration testing stage in your organisation? May you elaborate why you feel so?	
x	I would like to hear your thoughts on using **Integration Testing, PP, Refactoring, Unit Testing, Onsite Customer/Proxy, UAT, and CI** to address defects in SW integration and integration testing stage.	Can Integration Testing, PP, Refactoring, Unit Testing, Onsite Customer/Proxy, UAT, and CI be used to address defects during SW integration and integration testing stage in your organisation? May you elaborate why you feel so?	
xi	Have I missed any agile practice that could be useful to identify defects at this stage?	Which practice? Why do you think it could work at this stage?	
xii	What improvements or changes would you suggest at this stage?	May you elaborate why you feel so?	
	8. SW System Testing		
i	What are your thoughts on using **PP** to address defects in SW system testing?	Can PP be used to address defects during SW system testing? May you elaborate	
ii	What is your view on using **Unit Testing** to address defects in SW system testing?	Can Unit Testing be used to identify defects during SW system testing in your organisation? May you explain why you think so.	
iii	What is your opinion on using **Refactoring** to address defects in SW system testing?	Can Refactoring be used to address defects during SW system testing in your organisation? May you elaborate why you feel so?	
iv	I would like to hear your thoughts on using **UAT** to identify defects in SW system testing.	Can UAT be used to identify defects during SW system testing in your organisation? May you elaborate why you feel so?	
v	What are your thoughts on using **Onsite Customer/Proxy** to address defects in SW system testing?	Can Onsite Customer/Proxy be used to identify defects during SW system testing in your organisation? May you explain why you feel so?	
vi	What are your thoughts on the complementary use of **PP, Unit Testing, and Refactoring** to	Can PP, Unit Testing, and Refactoring be used together to address defects during SW system	

	address defects in SW system testing?	testing in your organisation? May you elaborate why you feel so?	
vii	What are your thoughts on the complementary implementation of **UAT and Onsite Customer/Proxy** to address defects in SW system testing?	Can UAT and Onsite Customer/Proxy be used together to identify defects during SW system testing in your organisation? May you elaborate why you feel so?	
viii	I would like to hear your thoughts on the complementary use of **Unit Testing, PP, Onsite Customer/Proxy, Refactoring, and UAT** to address defects in SW system testing.	Can Unit Testing, PP, Onsite Customer/Proxy, Refactoring, and UAT be used together to address defects during SW system testing in your organisation? May you elaborate why you feel so?	
ix	Have I missed any agile practice that could be useful to address defects at this stage?	Which practice? Why do you think it could work at this stage?	
x	What improvements or changes would you suggest at this stage?	May you elaborate why you feel so?	
	9. Acceptance Testing		
i	What are your thoughts on using **UAT** to address defects in acceptance testing stage?	Can UAT be used to identify defects during acceptance testing stage in your organisation? If not, may you elaborate	
ii	What is your view on using **PP** to address defects in acceptance testing stage?	Can PP be used to address defects during acceptance testing stage in your organisation? May you elaborate why you feel so?	
iii	What is your opinion on using **Onsite Customer/Proxy** to identify defects in acceptance testing stage?	Can your organisation use Onsite Customer/Proxy to identify defects during acceptance testing stage? May you elaborate why you feel so?	
iv	What are your thoughts on the complementary use of **UAT, PP, and Onsite Customer/Proxy** to address defects in acceptance testing stage?	Can UAT, PP, and Onsite Customer/Proxy be used together to address defects during acceptance testing stage in your organisation? May you elaborate why you feel so?	
vi	Have I missed any agile practice that could be useful to address defects at this stage?	Which practice? Why do you think it could work at this stage?	
vii	What improvements or changes would you suggest at this stage?	May you elaborate why you feel so?	
D	**AV4AD and Implementation in MDS development**		

i	What software development methodology do you currently follow?	Any specific reasons? If not agile, have you considered agile? Would you consider agile? If not, any specific reasons?	
ii	What are your views on the implementation of AV4AD and your current SW development methodology in addressing defects?	What impact would the implementation of AV4AD with your current development methodology have on the number and severity of defects? May you elaborate why you think so?	
iii	What is your opinion on implementing AV4AD and conformity to IEC62304?	Do you think it is aligned to IEC62304? If not, can you tell me why you feel that way?	
iv	Have I missed anything that could be good to have in AV4AD?	May you elaborate if we missed anything? Why do you feel it is important?	
v	What improvements or changes would you suggest that could enhance AV4AD.	May you elaborate why these changes are important?	
vi	What are your thoughts on the adoption and implementation of AV4AD in your organisation?	Could you explain why you feel that way?	
vii	What would you change to make AV4AD implementable in your organisation?	May you elaborate why you feel the change is necessary?	
viii	Which particular agile practices of AV4AD would you like to be promptly adopted in your organisation?	At which stage would you like the mentioned practices to be used in your organisation? Could you explain?	
ix	Would you like to use or recommend using AV4AD in your organisation?	Why? Why not? Would you explain?	
E	**Final Reflections**		
i	What would be your overall opinion on AV4AD?		
ii	Final comments		
	1. Anything else you would care to add?		
	2. Thank the respondent for their time and the valuable information.		
	3. Mention about how they can contact me		
	4. Ask if they would like to further be involved in the project.		

Appendix E: Semi-structured Interview Transcripts

E.1 Interview 1

Position: Lead Software Engineer

Experience in MDS: over 10 years

Interview date: 30 May 2024 10:00 to 11:30 am

Speaker	Transcript	Notes
MN	I would like really to thank you for participating in this discussion and your insights are really valuable to my research. So, really thanks.	
JG	Ok. No problem.	
MN	I am Misheck Nyirenda. I a m a researcher in the Regulated Software Research Center here at Dundalk Institute of Technology. And my research is about application of agile practices to address defects in medical device software development.	
JG	Ok.	
MN	Yeah. So, before we can start our discussion, would you allow me to record?	
JG	Yeah. Absolutely.	
MN	Alright. Thanks. Thanks, J, for allowing me to record the conversation and your responses are confidential. Yeah. Hmm.	
JG	Yeah, no problem. And yeah, and then sorry, just to introduce myself. I'm JG. I am a Lead Software Engineer at Company A.	
MN	Ok. Thanks, Jack. So, the format I propose is that I will ask you a question and then you provide your views and the experiences where applicable.	
JG	Umm no problem.	
MN	Yeah. So, Jack, would you want me to provide some explanation about the framework or you're OK for us to move to the next section where I'm asking the questions?	
JG	Uh, some brief overview of the framework would be beneficial, yes.	
MN	OK, OK. Alright. Thanks Jack. Ok so I'm sharing the screen. I hope you can see my screen.	
JG	Yeah, I can see it.	
MN	OK. Yes, so, I have developed the Agile-based Defect Addressing Framework based on the extensive research and the survey by medical device software developers. So I identified the agile practices that have been included in the framework and the uh, this framework focuses mainly on addressing defects when developing medical device software.	
JG	Mm-hmm.	

MN	So, the practice that you see in blue there, in the blue rectangles they have been designed to complement each other, and the framework provides the flexibility for medical device software developers to select a practice, a single practice at a phase or multiple practices at their phase and use them to address defects there or they have the third choice to select all of the practices at the phase. So, where all the practices have been selected, a best practice has been provided to say, you start with these followed by these. For example, if we go to the software requirements analysis phase there. If you select all the practices, there is model storming, behaviour driven development and then PP, TDD, onsite customer/proxy. So, if all these are supposed to be used at this particular stage, then I'm saying that first start with model storming and behaviour driven development. After that you implemented the PP, TDD and onsite Customer. So, that enhances the defect prevention at this particular stage.	
JG	Mm-hmm.	
MN	So, in this particular framework, as I said, the focus is on addressing defects by first preventing them in the early stages and then identifying those defects that may appear in later stages. So, the green symbol on the left side there represents defect prevention.	
JG	Umm.	
MN	So, the defect prevention essentially that takes place on the left side of this framework and the defect identification takes place later, on the right side of the framework.	
JG	Umm.	
MN	So, that's a brief about the framework. If you have any question that maybe you would want to ask. You can ask me.	
JG	Uh, yeah. So, the idea is that each stage as you go through the V and V model is its own agile sprints.	
MN	Yeah.	
JG	Yeah. So, just looking through the model storming, and on your left here as well you are following a test driven development style preparing your tests long before implementation.	
MN	Yeah.	
JG	Ok.	
MN	Yeah. Yeah.	
JG	Uh, yeah. It makes sense to me.	
MN	OK, alright.	
JG	Yeah. Umm.	
MN	OK. Yeah. So, uh, I would love if we could focus on the agile practice included at each stage and then when I ask you a question, you would then maybe provide some views	

	about how you feel about that practice at that phase and also your experiences in the organization about using them.	
JG	Mm-hmm.	
MN	So, starting with the requirements gathering stage, what are your thoughts about using user story, at this stage in terms of preventing defects.	
JG	At the? And yes.	
MN	Requirements gathering stage.	
JG	Requirements gathering, yeah.	
MN	So, you can feel free to discuss any of these at this stage.	
JG	Yeah. So, ultimately, I found it comes down to the quality of the user stories and the quality of translating from that to an actual ticket in your developers JIRA, GitHub. Whatever ticket management system you are using.	
MN	Umm. Umm.	
JG	Personally, I have been very interested in seeing how BDD approach would work using something like Gherkin syntax through the whole way would help minimize the amount of translations of requirements along the chain. Uh because that's typically where we have found defects arising where we might have had a very high level user story and it might get decomposed into a couple of requirements, but never enough.	
MN	OK, so, if I get you clearly, you are linking BDD at the requirements gathering stage. OK.	
JG	Yeah, I think it would. Yeah, and BDD, are you speaking of like BDD with something like Gherkin or a more general form of BDD?	
MN	Yes. So, this is where different stakeholders sit together to consider the requirement and refine it.	
JG	Umm. And. Yeah. So, there it's refined. Yeah. So, I'm not sure if I have answered your question or not.	
MN	OK. Yeah, I think you have. Yeah, you have answered the question. So, moving on and considering now the complementary use that has been indicated at this stage, for example, using user story and onsite customer together, how would you say about it?	
JG	OK.	
MN	Using user story and use case and onsite customer together.	
JG	OK. Umm yeah. So how would we then take the user story and help decompose it into requirements? Umm yeah. So, typically, I think we've done this through workshops and uh, proposing a set of requirements that come from that user story, verifying that with the customer and get them to understand the part that problems faced, really understand their uh. I think at this requirements gathering stage we may have done like stakeholder identification that will also help steer the conversation into, OK, well, this story. Let’s be	

	clear which stakeholders this is satisfying and who it's not, you know	
MN	Mm-hmm.	
JG	And typically, we used to gather then to decompose that into I guess system requirements or more targeted user stories at the stage.	
MN	OK.	
JG	Umm, yeah. So, I think probably the issues we might have bumped into in the past would have been things like getting the right person to talk through these problems, you know, so if we have uh perhaps, like a user story, use cases or user stories for a clinician, but we might not be able to get access to that clinician because of their limited schedule. So, we have somebody making a best attempt at that, you know, and so there's sometimes where some of the defects may not be identified umm at that time.	
MN	Hmm. OK.	
JG	You know it's a constant struggle in some of these projects just getting those correct people. Umm.	
MN	OK. But in the case where you have the correct customer, have you found these practices to be useful in terms of preventing defects?	
JG	Yes, you know, but yeah. Yeah, but uh, it prevents some.	
MN	OK.	
JG	I'm not sure the approaches were taken as a too difficult for. Still just difficult for, you know, an average person to understand.	
MN	Umm.	
JG	Big through the built system and how it could be used and the places where problems gonna rise. You know, we've, uh, people tend to focus on the happy path and the successful path through without thinking OK well, what if you've a few 30 times you went into this that you should have, you know. On the edge case is that I think developers tend to uh focus on because it's where we have to fix that so.	
MN	OK, OK. Mm-hmm.	
JG	Uh, so they get pretty edge boundary analysis, and our edge case analysis might not be as solid yeah.	
MN	OK. Alright. Thanks Jack. Uh, now moving to the software requirements analysis stage.	
JG	Umm.	
MN	Uh, what would be your views on Model Storming, PP, TDD and also onsite customer to prevent defects at this stage.	
JG	Personally, on uh, in previous jobs on like nonregulated software I found BDD very useful for, and in particular just to frame the conversation and keep things to a regular	

	language that we can help pull out those types of boundary cases from people.	
MN	Umm.	
JG	In terms of model storming, we are investing a lot more in model-based systems engineering and so certainly something we have been approaching, it's a bit of a growing interest in BDD.	
MN	Umm.	
JG	But in terms of our like the model storming we have done before, might have not been as formal, making more loose architectural diagrams and walking through them in the workshops on how problems could arise.	
MN	OK. Hmm.	
JG	So, yeah, I can say we probably don't have great insights on it just yet, you know, but it's somewhere we are absolutely trying to improve on.	
MN	So, you have talked about two mainly on the nonregulated side, uh, would you think that they would work in the regulated part like in your organisation, would they be useful to address defects?	
JG	Sorry, they work in the nonregulated side; would they help with the regulated side is that the question or?	
MN	Yes, you have mentioned, if I got you correctly, you have mentioned that from your previous experiences in the non-regretted field you used them and you found them to be very effective.	
JG	Yeah. Umm.	
MN	So, I'm asking in the regulated area now where you are, would you think that they would be useful to address or to prevent defects?	
JG	Absolutely. Yeah. I mean the concept still apply. You know you are thinking of the problem in terms of actors or personas, and you know things like design thinking.	
MN	Ok. Ok.	
JG	You know, they are great tools that I think in the regulated world argue that one tool is enough and think that sometimes it can be a little culturally a little slow to move.	
MN	Umm.	
JG	So yeah, so certainly they have been useful in my experience and yeah.	
MN	OK, OK. Yeah. So, you have only spoken about model storming and BDD. Would you also speak about PP and TDD as well as onsite customer?	
JG	Umm. Uh, yeah, I'm. So, I'm a big fan personally of TDD. I find it can be difficult to get it done in practice and it's usually, you know, after you have some requirements, the commercial pressures tend to make like every software project I've been on has started too early.	

MN	Hmm.	
JG	Umm that seems to be the case in industry. Commercial realities want people to get pushed. We've deadlines to hit, get people working and things like TDD tend to get pushed down the road, you know? So, we can just write the test cases later, you know, defeating the whole purpose of TDD.	
MN	Mm-hmm. Ok	
JG	Yeah, I'm a big fan it. I think it would help a lot too, especially in this requirement stage to again that boundary analysis really that 5% of problems that can consume 70% of your time, you know. It's and certainly be thought of seeing it implemented, but personally I haven't seen it yet, a very successful run of it in the past. So yeah, it's really that kind of and you have centre of requirements, and you are defining an architecture that I think commercial pressures tend to build up at that point and.	
MN	Hmm, OK. Umm. Hmm.	
JG	People want to see something built, you know, unfortunately.	
MN	OK, umm and PP, what about PP?	
JG	And a pair programming. Yeah, a fun. In the context of it it's just what in terms of your requirements like pair programming, during requirements analysis, or I suppose most would associate pair programming with the development stage itself.	
MN	Yeah, but if it was used at this stage, would it be helpful to address defects or to prevent defects?	
JG	Yeah, but when? Perhaps maybe lower when you are building the architecture to have that oversight. And would this be pair programming with? Is it still with the customer or with?	
MN	Yeah. So, the proposal there is that you use that with the customer as well.	
JG	OK, OK. Yeah, it's. I would be interested to see it happen, but I yeah, I think typically the people we are dealing with in the medical and pharma world you are starting to get too low level in which getting that kind of time at this point, if you had a strong product owner, perhaps it would be a good person to have in there as the as the customer representative in that case.	
MN	OK. Umm. Yeah, yeah.	
JG	But you know you're not probably never going to get like a clinician to sit with you and do the pair programming, and that fast time of those people is very difficult to get.	
MN	OK. OK. OK. Umm.	
JG	Yeah. I mean, it certainly would be helpful, you know, and anything design wise you can do earlier the better you know, but uh, yeah, it's just I don't.	
MN	Hmm.	

JG	I'm not sure in reality how likely would be.	
MN	OK. Alright. So, if I get you clearly the main issue of not maybe using it often at this particular stage is to get the right customer to support it.	
JG	Yeah.	
MN	Ok.	
JG	Umm, yeah. That's usually, yeah. Yeah should.	
MN	OK, alright. So, what about the complementary use of these practices at this stage? Like we have talked about, maybe using them singularly and then now if you try to complement, which compliment would you think would prevent defects?	
JG	Umm. I'm. Yeah, the model storming and BDD makes sense together. Umm.	
MN	Mm-hmm.	
JG	Especially if you have a good visual model and, yeah that makes a lot of sense for me and pair programming, TDD together.	
MN	Hmm.	
JG	One second, yeah. Like theoretically that PP, TDD and onsite customer makes a lot of sense. You know, like it sounds like a good idea to aim to.	
MN	Ok.	
JG	Yeah, it's just, yeah, in my experience it's just been difficult to just get that resource, you know. But no, I think the ones you have do make a lot of sense yeah, to combine, yeah. Yeah	
MN	Alright. Let's move to the software architectural design stage. So, what are your thoughts about using model storming, onsite customer, integration testing, and simple design at this stage? What would be your views or opinions?	
JG	In terms of integration testing do you mean at this stage like go and test driven design and writing the integration tests at this point?	
MN	Yeah.	
JG	Aah. Ok.	
MN	So, it could take that form, but also if you involve integration testers as at this stage, would they be able to point out some defects that would maybe appear later if you don't think of involving them there?	
JG	I actually, yeah, I was thinking back to some projects. Yeah, we had defects that arose, and you know, design defects that arose during the implementation phase that could have been easily avoided by some integration testing at that point.	
MN	Umm.	
JG	You know we were integrating with large Phillips digital health platform; you know and the flaw some of the basic connection logic had not been property tested and was left	

	to further down the road to the implementation that uh that caused issues.	
MN	Mm-hmm.	
JG	So yeah, some integration testing there makes sense. The model storming, I am not sure like at that point should you be just working off the requirements. Certainly, could help reinforce, but I'd. Yeah, my instinct would be at that point to leave the model behind, but yeah, I will consider this approach, yeah.	
MN	Umm. Mm-hmm. OK. Mm-hmm. And what about the simple design?	
JG	What do you mean by simple design? The block work diagrams or?	
MN	Yeah. So, making the actual design to be simple for easier modifications later.	
JG	OK, OK. Yeah, yeah, some type of high level design there. Yeah, makes sense. And then again Integration testing and onsite customer. Uh, yeah, makes sense there. Yeah. So, the only one in this section I wouldn't be sure of is just the model storming.	
MN	Mm-hmm.	
JG	At what point, do you leave it behind or. Are you? Do you constantly reinforce to check in that you're always aligning with? Or so you are always validating Uh, your requirements.	
MN	Mm-hmm. Ok. Alright. And what about the complementary use of these practices in terms of preventing defects.	
JG	Yeah. Yeah. At this point, uh mmm they make sense together, customers love seeing that simple design that they have something to latch on to that they can, you know, go back to their stakeholders and say look here we have a design they work with their tech teams.	
MN	Hmm.	
JG	Any kind of integration test of this stage is I would be surprised to see anybody implement this at this stage.	
MN	Mm-hmm.	
JG	Certainly, the design is a good use for discussion to help identify defects. If with those customers getting the right customer proxy, you know. So, for example, when we were integrating with uh, like that Phillips digital health platform, that we can give it to a customer's architect and say look, is this going to integrate? Is the DNA going to the correct place for use with your external tools?	
MN	Mm-Hmm.	
JG	Like the design is enough to get the conversation started, but probably not detailed enough for really defect identification. So, you are trying to so. So yeah, for example, if you are using something like, file that.	
MN	OK. Umm	

JG	Customer gonna be very happy with your high level structure looks great. The data is going to the right place, but you know a lot of the defects are going to come up because of the structures of the data you are using, and I guess that's in the next stage, the detailed design, probably you identify more defects at that stage, really, in my experience. But yeah, sorry, in terms of complementing, yeah, they do.	
MN	OK.	
JG	Again, the question mark would be at the model storming, at this point, yeah.	
MN	OK, OK. Yeah, you have already hinted on the next stage, so let's move there (SW Detailed Design).	
JG	Yep.	
MN	And yeah, what would be your views on these practices. Model storming, onsite customer, integration testing, and simple design at the detailed design stage in terms of preventing defects.	
JG	Yes. Yeah. Like the simple design, I think at this stage probably needs to evolve to have some good detail before you get to implementation. The things like your contract testing, you know, does your API actually give me the information I need. You know that kind of scenario would want to be ironed out at this stage.	
MN	Mm-hmm.	
JG	So, the simple design probably needs to be evolved to be something more thorough.	
MN	Umm.	
JG	To add, and there's just a story from my previous experience. And again, the same question with the model storming.	
MN	Yeah, yeah.	
JG	I suppose my worries with the model storming is that it's constantly re-examining the opening requirements that you are opening the chance for requirements to change, and that's right up there.	
MN	Mm-hmm.	
JG	So yeah, I just, I would have a question mark on that (model storming). That's all. But yes, the simple design I think needs to be evolved. The compliments look well together, but simple design I think probably does need to be something more thorough more ready for a developer to pick up. And you know, if I didn't have a ticket, they could still probably implement it.	
MN	OK, OK. Umm. Alright. Anything more to say on this stage in terms of agile practices?	
JG	No, I don't think so. I think at this point, typically in our processes, you know the commercial pressures have reached and the implementation is probably started in other features. Yeah, you know, it's. So, by the time we have gotten to our first bit of detailed design with probably got developers	

	starting bootstrapping the app. You know getting the framework in place and all that you know. So, a lot of these couple stages tend to happen more parallel than they should.	
MN	Ok.	
JG	Uh, you know, that's just commercial realities. People have deadlines that so.	
MN	OK?	
JG	And so, then I guess we can start unit implementation.	
MN	Yeah, let's move there, yeah.	
JG	Actually, the coding standards. Gosh, I was gonna say you can move that further up, but really that should be defined prior to this development at all.	
MN	OK. Yes, so you are saying to move it to the detail designed. OK.	
JG	Mm-hmm and yeah, I mean, it probably should be part of somewhat part of the Quality Management System (QMS). I did, but anyway.	
MN	Hmm.	
JG	Maybe that's in reality, probably not. It's probably wouldn't be ironed out at that stage and. But yeah, at unit implementation, yeah, you imagine there would be implementing any of the risk mitigation or defect prevention, mitigations we found earlier.	
MN	Hmm yeah.	
JG	Umm that we find in some they'll be fixing them, so pair programming. TDD makes a lot of sense together.	
MN	OK.	
JG	In terms of the onsite customer/proxy. You know, usually we would have some project owner that would take ownership of that and manage any of this, the low level detailed requirements, you know like the validation of a field on a page, you know something very small doesn't really need to be in the discussion at high level.	
MN	OK. OK. Hmm. So, you find this practice (onsite customer/proxy) also can prevent defects at this stage.	
JG	It's in terms of preventing defects, it's typically requests require low level requirements that you would have identified there like, sorry, I am gonna take an example. A login page that somebody needs to log in with an email address. You know, you identified that there's defects account arise from that, but you won't really get into the mitigation measures. You know it's like you can figure that out later.	
MN	Hmm.	
JG	That's fine. So, you get that at detailed design and then when you get to unit implementation that, between the project owner and the developers, they will figure out what's the best solution for that.	
MN	OK. Umm. OK.	

JG	So, again, like there can be defects that arise from that if your software is interacting with a different system and suddenly you've missed that the external system has one set of validation rules and you've decided on a different set.	
MN	Umm. Umm.	
JG	Umm yeah, there's absolutely chances for new defects to come in there that perhaps could have been prevented before.	
MN	Yeah. OK.	
JG	But again, it's always that pressure of the commercial side of trying to not become waterfall again.	
MN	Umm hmm.	
JG	Yeah, yeah. You know, talking to colleagues is that any projects that they would have done to that level of details in the design phase, had trouble just getting started and it can kill the project before it starts.	
MN	Hmm.	
JG	So, I think there's sort of an acceptable.	
MN	OK	
JG	There is I guess typically an accepted amount of defects that are going to come up in the implementation phase. Maybe that could be identified earlier. Yeah. Say we know there can be defects here.	
MN	Yeah.	
JG	Here is a rough scale of how many equal them out of this, and that should be used probably as a metric for moving on to the next stage.	
MN	Hmm.	
JG	It's like, OK, if this is actually quite high risk, there's quite a lot of risks that come out of this.	
MN	Hmm.	
JG	We probably shouldn't go to the implementation and, but yeah.	
MN	Hmm. Mm-hmm. OK. Yes, yes. So, you have spoken about TDD, PP.	
JG	Umm. Yeah.	
MN	You haven't spoken about code ownership and model storming as well.	
JG	Yeah, so here, the model storming it's one that I again, question mark of. Would there be no room if people are looking at the model and not the requirements that were derived from the model, is there the chance you are increasing the misinterpretation of what actually needs to be implemented to sort of?	
MN	Mm-hmm. OK. OK, OK.	
JG	But again, I totally see the value of it as a tool to constantly reinforce the problem you are trying to solve with the software.	
MN	Yeah, yeah.	

JG	So, yeah, I'm on the fence (undecided) currently on the implementation. Uh coding standards and yeah absolutely should be there.	
MN	OK, OK. Hmm.	
JG	Code ownership uh mmm yeah. Typically, yeah, these. I am a fun of it, like developers on these types of projects, you don't get to see the type of churn that you would get in, you know, a SaaS project in the nonregulated field.	
MN	Mm-hmm.	
JG	So, that code ownership is great to have that institutional knowledge being built up and so would you have someone you can depend on, because especially as some of these systems get so complex.	
MN	OK.	
JG	Yes, they make sense. Yeah, they all make sense together. So, in this case the onsite customer/proxy in our version would be a product owner, yeah.	
MN	Yeah. Umm OK.	
JG	And then typically then the small release that might be a different customer representative, you know. So, perhaps the project manager on their side, or someone who's a bit further up the corporate hierarchy	
MN	OK. Mm-hmm. OK, OK. Yes. So, you have hinted on the small release there, uh? Does it help to identify defects during this stage (SW unit implementation)?	
JG	Umm, so unit testing, small release, refactoring Umm. Yeah. Yeah, these all make sense to me.	
MN	Hmm.	
JG	Yeah, yeah, it all makes sense. All makes sense together.	
MN	OK. And then code review? For identifying defects at this stage?	
JG	Umm. Yeah, I would uh. Are you saying, these are separate stages to prevent, identify, identify, identify or you are saying it's like gates that should be passed through before the code gets into the main release or?	
MN	Yes.	
JG	OK, OK. So, typically it’s like your unit testing, code review, CI is all really bundled together. You know, I wouldn't really see them as separate things.	
MN	OK, OK.	
JG	I can tell you of a typical process. Someone opens up a pull request or merge request and the unit tests are run at the same time, the code reviews happening, it's not quite sequential.	
MN	Umm, OK. So, you're saying that it would be more effective if they are like complementary these three (unit testing code review, and CI).	

JG	Yeah. Yeah, I think so. Because typically you look at it holistically like your unit tests are OK. Umm, but on their own they are not necessarily valuable because you can have 100 tests that don't really tell you how well it works.	
MN	OK.	
JG	So, it's that (unit testing) paired with code review, it's far more valuable I'd find.	
MN	Umm.	
JG	And then in the set for the CI bit, what do you mean by that? Is that static code analysis or?	
MN	Continuous integration. So, where you're adding the code that has passed these unit tests and code review into a code base.	
JG	Uh. OK. So, yeah, I guess the CI and small release are kind of the same thing of doing a build.	
MN	OK.	
JG	Yeah.	
MN	So, you are suggesting that these two can be done complementary to identify defects.	
JG	Yeah, I think so. So, I would consider probably the CI and small release the same thing and then code review is complementary.	
MN	OK, OK.	
JG	So those 3 defect identifications (*meaning the separation of code review, unit testing and code review in AV4AD*) are probably one in my experience.	
MN	Umm.	
JG	But and at last the defects aren't there so.	
MN	Mm-hmm. OK. Umm, I think refactoring. You haven't really discussed it.	
JG	Umm. Ooh refactoring uh. Yeah. So, I guess I am not sure what you mean here using it because I guess refactoring is almost reimplementing, so you are saying just coming back around the loop.	
MN	Mm-hmm. Yeah. So, uh, the idea is that you have a unit of code that has been developed and once testing you find that it doesn't work as what you require. So, would be maybe refactored to make sure that it conforms to what you need.	
JG	Umm.	
MN	So, would it be possible or is it OK to include refactoring there to identify defects or to correct defects?	
JG	Yeah, I mean, I guess. The refactoring is the defect prevention, you know, so it's almost looping back to itself. Umm.	
MN	Umm. Yeah, yeah.	
JG	So, the refactoring really is to prevention, rather than identification.	
MN	OK, alright. Umm, uh, let's move to the software unit testing stage. What are your opinions on using unit testing to	

	identify defects and onsite customer to identify defects and also using PP and refactoring at this stage?	
JG	So, at this do you mean the testing this, so do you know, I guess traditional unit testing. You take a very small function, and you test it, whereas yeah, this you are talking more about the software unit as we have defined.	
MN	Yes, yes.	
JG	Yeah, well, you know it could be a whole collection of features.	
MN	Yeah.	
JG	Yes. So, they are very useful uh in identification. Whether manual or automated or whatever, where you have it.	
MN	Umm.	
JG	The customer here proxy, that for us, that's like that product owner, that's that low level. That very low level requirements clarification, you know. Is this actually a defect, you know. It's almost always the requirements, if we have encountered an issue requirement doesn't really talk about it. So, engage with that product owner or customer proxy for it.	
MN	OK.	
JG	So, yeah, it can be very, very useful. Very useful cause umm.	
MN	Mm-hmm.	
JG	Certainly, helps to refine their requirements. The pair programming, I guess in this case would typically be a tester and the I have multiple testers.	
MN	Mm-hmm.	
JG	So, sometimes a tester working with a developer umm, which can be helpful to and.	
MN	Yeah, yeah.	
JG	Sometimes just complexity of the system can be reduced by the input of the person who implemented it can be useful. I'm not sure how strictly correct. That would be because I guess there's room for bias to come in there.	
MN	Mm-hmm.	
JG	But yeah, it would be useful if it was pair programming with multiple testers would be even better, you know. Umm, you kind of want that one biased locking it.	
MN	Mm-hmm. Ok	
JG	Yeah, they are all complementary for defects identification.	
MN	Hmm, OK. Alright, let's move to the software integration and integration testing. Uh, can you share some views on the use of use of user acceptance testing, onsite customer/proxy, integration testing, pair programming, refactoring and unit testing, as well as continuous integration at this stage to identify defects.	

JG	Yeah. So, at this stage integration testing is vital. Really helps you identify more system level defects or system level design defects.	
MN	Umm. Hmm.	
JG	So, the unit testing, I'm not sure because at this point you already know, if you have gotten here, your unit tests have passed. So, if you have missed something in your unit testing, looking back at them probably isn't going to help. It might help you like to isolate the issue. It's like, OK, what? It might help you identify that a defect is solely an integration issue which will help you identify that it's probably an issue with the detailed design or the requirements.	
MN	OK. Mm-hmm.	
JG	But uh. So, it's OK as a reference point, but not that valuable to redo the unit tests. Look at the unit tests and really your focus would be on that integration, whether that's you have built an app connecting to a server and you are doing a full end to end test, you know. Just with that server and your app.	
MN	OK. Mm-hmm.	
JG	Pair programming yes would be great to have with it. And then refactoring at this stage, yeah. I mean, if it needs to be done, it needs to be done. And CI something. Yeah. I'm not totally sure if you mean refactoring and CI here. So, this point saying make another build and then do another integration test with a new build with one with a defect prevented or?	
MN	Umm. Yeah. So, the thinking is if a defect is identified in the Unit that you are integrating, that (unit) should be refactored and then put in the build again.	
JG	Mm-hmm, then OK, yeah. And then, yeah, it's absolutely yeah, complementary in that case.	
MN	Mm-hmm.	
JG	And yeah, I think you are correct then not to have that customer or proxy then because at this stage you should know how this is meant to integrate with the system. So, the customer doesn't necessarily need to be there for that, because at this point you should have all the information you require.	
MN	Ok. Umm.	
JG	Yeah, you should. Yeah. And then the next thing, UAT.	
MN	Umm.	
JG	Yeah, absolutely. Customer there, if there is any defects identified, it's likely outside of the boundary of your team's knowledge, so you know you've integrated with system.	
MN	Hmm.	
JG	For example, you know you are saving vitals information and at this stage, the customer may be able to identify that	

	actually that information is being saved in a way that doesn't propagate to their other systems.	
MN	Hmm. Umm.	
JG	And it's like, well at this stage you would never find that with the knowledge you have now, only the customer and their UAT is really identifying that and should have been identified earlier really in the requirements.	
MN	Hmm. mm hmm.	
JG	But it's difficult at that stage when you are not getting that low level to identify these kinds of bugs that pop up the defects.	
MN	OK. Alright. And so, anything more to say on this stage?	
JG	Umm. No, I don't think so.	
MN	OK uh, then we can move to the software system testing stage.	
JG	That's right.	
MN	I would like to hear your views on using PP, unit testing, refactoring, user acceptance testing and onsite customer/proxy at the software system testing stage.	
JG	Yeah. At this stage, really the unit testing I think is really out of scope at this stage	
MN	Mm-hmm.	
JG	Because you have gone this far, the units should be working, so really probably focus should be more on that end to end testing that the system behaves as designed, you know, your expected behaviour for the system is working.	
MN	OK. Umm.	
JG	So, yeah, I wouldn't reset unit testing more. And yeah, probably end to end would be the way I put it.	
MN	Mm-hmm. Ok.	
JG	Yeah. Again, pair programming can work great at any point in the stack really. And then refactoring, if needs be and you know you build, publish whatever needs to be done, yeah.	
MN	Ok. And what about user acceptance testing?	
JG	Umm. Yeah, absolutely. Yeah, that is probably what I'm saying as end to end testing. Really, it's like user acceptance testing, that they should at this point be able to identify if the system is behaving as expected.	
MN	Mm-hmm. OK.	
JG	And then I guess we got to the acceptance testing which the customer user acceptance test would identify is it actually, uh, solving their needs, the users' needs?	
MN	Mm-hmm.	
JG	And validate that against it and if you know if there's defects identified here. That's yeah, it so far back down the chain to fix it. But, umm, realistically, there should be nothing coming up here.	
MN	Umm.	

JG	And like this is probably the point at which, I think this is the probably the big argument that model storming should be a continual part of the of the downward side of the V-Model that you should really be getting to acceptance testing and having your failures of your design.	
MN	Umm. Hmm. Ok	
JG	Always happens but, yeah, it's certainly question I'd like to see how far would model storming all the way down continually through the implementation process, identify the defects that would have come up at acceptance testing.	
MN	Umm so, you are saying model storming should be continued through which stage?	
JG	Down to the implementation or throughout the process, you know, whichever.	
MN	Uh, OK.	
JG	Probably to the implementation, you know, that's when you are implementing. When you are doing small releases and those.	
MN	Mm-hmm.	
JG	At the implementation stage, is the highest chance of ad hoc design decisions being made. So, the model storming at that point would certainly help to reinforce and prevent changes that veer away from the model and the requirements.	
MN	OK.	
JG	So, it's, you know, it's your Bible, your source of truth here. You are keeping that mission statement top and center.	
MN	Mm-hmm.	
JG	Yeah. Yeah, so.	
MN	Alright. So, going back to the software system testing. What changes would you propose at this stage? In terms of agile practices that would work there to identify defects. What would you propose as changes or improvements there?	
JG	System testing. I would think that unit testing isn't really applicable there.	
MN	Umm OK.	
JG	It's really that UAT and end to end testing that is valuable there. At this point you have done extensive unit testing, you know the individual parts “work” and you have done integration testing which is always going to be a little bit limited, you know.	
MN	Umm. OK. Hmm. Umm.	
JG	You should theoretically know when from A to B it works from B to C it works, but you don't know from A to B to C does work?	
MN	Hmm.	
JG	And so that's really what you wanna solve and that’s what system testing is from A to B to C should work. I know A to B works and B to C works, so what can arise there is an issue.	

MN	Yeah.	
JG	And. Yeah. Yeah. So, I am trying to think of a problem that could come up like that.	
MN	Mm-hmm.	
JG	Yeah, it's probably like bad data, bad contracts and API's, like your taxonomy is slightly different. Performance requirements perhaps could the thing that arises here that actually, individually, your constituent parts work well on their own, but as part of a larger system actually the latency is unworkable, you know. It's taken 30 seconds for a request to propagate through your system, and that's now what you're gonna fill for your high level performance requirements or user needs.	
MN	Mm-hmm. Hmm. OK.	
JG	Umm, so, I think that's typically kind of the things probably you find.	
MN	Umm.	
JG	So, yeah, so like unit tests not really applicable at this stage, but yeah that end to end testing. I guess what we call our system testing.	
MN	OK, alright. Hmm. And moving to the acceptance testing stage.	
JG	Yep.	
MN	What are your views on the use of user acceptance testing, pair programming, and onsite customer to identify defects at this stage.	
JG	Yeah, it's a very valuable umm. Given uh. I think I'm just trying to understand what types of defects you would even find in this stage.	
MN	Hmm.	
JG	And I imagine they would be mostly like performance related requirements and then kind of functional.	
MN	Hmm.	
JG	It would be a big mess if you find a functional defect here at this point.	
MN	Hmm.	
JG	But typically, yeah. Like if you are doing that acceptance testing in a more or less simulated environment. Your system testing might happen in a simulated environment that is probably the best case scenario.	
MN	OK.	
JG	So, if the acceptance testing happens in a less controlled environment, more variables, would have to go wrong, you know, spotty Wi-Fi.	
MN	Mm-hmm.	
JG	Yeah. Makes sense to have that user acceptance testing there and customer proxy there.	
MN	Hmm, PP, pair programming?	

JG	Yeah, I guess just identifying who the pairs are in this case. Is it technical representatives and a customer? Or is it multiple customers? Uh yeah. I think it's just probably for each of these is “who is in the pairing” question. But I think that would solidify some of the definitions of this model, this framework.	
MN	Umm. Mm-hmm. Come again?	
JG	Uh, so yeah, for each of the pair programming if you could identify the “Who is the pair?” we are talking about in each scenario.	
MN	Hmm, OK.	
JG	That would help?	
MN	OK, alright. And so, we have at least talked about practices at each phase, but I would want to ask if there are practices that you feel are missing at each or any phase within the framework?	
JG	Umm.	
MN	That would either prevent defects or identify defects. So, we can start by looking at maybe requirements gathering and then you can say which practices would work there to prevent defects.	
JG	OK. I think going down the left I think you have identified most of them here. The requirements gathering you are model storming, creating tests. Yeah. All great practices. Probably the one that I found in the past that is at the detailed design, having the integration design ironed out at that stage would prevent a lot of defects and it’s something that can go I guess, get overlooked is how exactly your software units are integrating with other systems.	
MN	Mm-hmm.	
JG	If the defects for that can be identified too late that it becomes quite expensive than to go back and refactor.	
MN	OK. Umm.	
JG	Umm. So yeah, so at that detailed design some detailed integration design.	
MN	OK, uh. What practice should that be?	
JG	It's detailed integration design.	
MN	OK.	
JG	Like you, say, contract testing. Sometimes it's called.	
MN	Umm.	
JG	So, like before you start your implementation, you should know the data schemas that are expected and are not and be extremely rigid on those. Because it just becomes so expensive to roll those back.	
MN	Mm-hmm. OK.	
JG	Umm, yeah, it's one that's yeah been a pain point a couple of times.	
MN	Hmm.	

JG	It could be something so simple as, you know, having the correct response codes coming back. I am trying to submit a form, but my software needs to change its behaviour based on your response but at that point if I don't have the details of that response that kicked us in the past where it actually wasn't what we expected and we would go back around, and the other system had to make changes that caused delays.	
MN	Umm.	
JG	So, yeah, having that up front. That's been a real pain point in multiple projects.	
MN	OK.	
JG	Then everything else. I think there is nothing missing. No, I think covered it.	
MN	Umm, which practice would you say that maybe could be changed or could not be effective to address defects at a particular stage?	
JG	For addressing defects uh. OK. Let's say refactoring. No. But I think they are all useful. It's just in what you've identified.	
MN	OK.	
JG	I think it's always just if they actually get implemented when deadlines are approaching, you know, and that's always the issue.	
MN	OK. Hmm. OK. So, which software development methodology are you following currently in your organization?	
JG	It's aah I would say it's quite similar to this V and V model, but uh at the bottom of the model is where we consider ourselves as agile and the rest of the model is far more traditional waterfall.	
MN	Mm-hmm.	
JG	So, for example, we are probably trying to move to more agile the entire way, but typically it's, you know, a lot more design work up front, to an acceptable point, and then we hit the implementation where it's quite agile and trying to uh.	
MN	Mm-hmm. So, would i be right to say you sort of hybrid?	
JG	Yeah. Hybrid (agile and waterfall) would be yeah.	
MN	OK, OK.	
JG	Yeah.	
MN	Alright. So, I want to ask if implementing the current model and this framework, would it be useful in terms of addressing defects? I mean implementing your current methodology and the Agile-based Defect Addressing Framework. Would that be useful to address defects.	
JG	Yeah. Yeah, it would. It would. Particularly, I think at the requirements stage using BDD and TDD earlier would certainly be very beneficial to identify those defects earlier.	
MN	Hmm. OK.	
JG	Yeah.	

MN	Right. And so, now looking at the overall framework, the Agile-based Defect Addressing Framework. Would you adopt it in your organization for medical device software development and specifically for looking at the addressing defects?	
JG	Uh, yeah, I think we would, again with those changes, you know, perhaps, you know. At system testing, we are not really looking at unit tests and at the detailed design we are looking at a more thorough integration design or contract design.	
MN	OK. Mm-hmm.	
JG	But no, yeah, this would align with I think the values we have going forward and yeah, so it's yeah, it would be interesting.	
MN	OK. OK. Any thoughts so far that maybe I haven't touched on throughout our discussion that you would really want to point out in relation to the framework.	
JG	Perhaps what I can say is just that the identifying is what some of the realities of development where the architectural design, nothing ever happens nicely phased so you know that the architectural design is still going while implementation is going. And so, you know the implementation of software unit A is happening, but the architecture and the detailed design for C, D & E are still in flux and still happening. But the implementation of A & B can go fine because they are low risk, mostly defined items.	
MN	OK, OK. Hmm. Hmm. Hmm.	
JG	So, yeah, I don't know how to visually represent that. If it's like arrows, going the whole way down and all the time.	
MN	Hmm.	
JG	But in reality, a lot of time it's always feeding back up towards requirements.	
MN	Ok.	
JG	And I have never been on a project where it's been properly phased. You know it's always feeding back up to the master requirements.	
MN	Hmm. Hmm.	
JG	Uh, yeah. So, like the requirements after your requirements analysis, you have probably got 70 to 80% of your requirements, but there are still gonna be more. They are gonna come up and feedback into it and be part of that master list at the end.	
MN	Umm. Yes. So that's one uh key aspect of this framework that it gives you the flexibility to go back to a stage to correct a defect that maybe found in a later stage so that really prevents from costly rework which would be incurred later on.	
JG	OK. Yeah. Yeah. Yeah. So that's probably just the visual representation. It's always hard how to put it on a graphic.	

MN	OK, so, what would you suggest to improve? Like you have pointed out, the visual representation there, what would you suggest could improve?	
JG	Yeah. Hmm. It's yeah, trying to think. It's always like parallel streams that go back up and then it's almost like the bottom of the V-Model is almost like there's multiple of them, whereas detailed design for software item one happening, then it goes to implementation that goes to unit testing and at the same time software unit B is being designed but A is being implemented.	
MN	Umm.	
JG	So, it's trying to I suppose to visualise that, the parallelism that happens in reality which you know is fine and that usually is umm.	
MN	Umm. OK.	
JG	Uh, so yeah, I think the bottom of the V-Model those bottom 3 phases tend to be a little more fluid.	
MN	Mm-hmm.	
JG	Or yeah, it's almost parallel, fluid maybe the wrong term, but yeah.	
MN	OK. So, one thing that I would ask is about the standards for medical device software development.	
JG	Mm-hmm.	
MN	Yeah, like the IEC62304.	
JG	Umm.	
MN	Yeah. So, uh, are you familiar with it or?	
JG	Yah, yah. We implement it here.	
MN	OK, so looking at that standard and referring to this framework, uh, what would you say?	
JG	Yeah. That's where we have adopted a similar model ourselves internally, and we do adhere to 62304. They are compatible.	
MN	Hmm. Hmm.	
JG	Uh. Your software development plans are in place at this point. You should have I would say like before, you probably have a lot of those IEC2304 questions defined before you even start on this. You know, like your branching strategy and you know, releases all that.	
MN	Umm. OK.	
JG	So, it's complementary. So, I am not sure if you need or are you thinking of like including this in this framework or?	
MN	Yes. So, if maybe your organization adopted this framework, would you still conform to that standard using this framework?	
JG	I believe so. Yeah. Yeah. It’s similar to what we have now. So, I would, I would think so, yeah.	
MN	Mm-hmm. OK. What other comments do you have for me?	

JG	No, I think. Yeah, I think the main question I probably have with this is just how model storming all the way down to implementation works in practice?	
MN	Mm-hmm.	
JG	So, it's that question. If you are constantly storming the model and bypassing the requirements, derived from the model, does that impact if the delivered software conforms the requirements? So, like, if you have got your model, you make requirements and then when you are implementing you always keep looking back at the model, are you still conforming to the requirements you derived?	
MN	Umm. Mm-hmm.	
JG	Or does it actually strengthen and help you refine those requirements so that I think that's probably the big question I am curious of out of this.	
MN	Yes. So, it helps to clarify the requirement.	
JG	Yeah, I guess my concern always is that by constantly looking back at the model you have now two sources of truth. There are requirements and the model, so, I think fundamentally I think it sounds like a good idea. But yeah, it's just in practice I am curious how well it does work. It's a question we're asking internally as well.	
MN	Umm. OK.	
JG	We are trying to figure out how much model storming fits into our process.	
MN	OK alright. Umm. Hmm.	
JG	So, previously we have, you know had the model, derived requirements and then work off the requirements. So, you don't have that same reinforcement.	
MN	OK, so, the implementation was like first have a model and then you generate the requirements.	
JG	Yeah.	
MN	OK. And you found that to be like in terms of relating to defects, was it helpful that way or you do want it to have this, like go hand in hand?	
JG	I think we haven't done the other way just yet in a full production of possibly going back to the model to reinforce the requirements. So, I can't compare just yet, but like we still have, it's always there in the quality of the requirements that you have derived.	
MN	OK. Hmm.	
JG	Umm. Perhaps quality could be better if we were referring to the model. I guess I am just always concerned it could also cause misinterpretation of requirements, or you know requirements aren't updated after the fact or I'm very.	
MN	Mm-hmm.	
JG	Is it something we are investigating. I am very interested myself as well.	

MN	OK. So, what about the way it is proposed to say you start with the requirements and then from the requirements you generate a model. Which would help you to clearly understand the requirements?	
JG	I think the model first makes sense. Because the customer engagement is a lot better with the model that people can see. Stakeholders identified, different actors in a system. You can pull better conversations out of customer with that visual tool, we found.	
MN	Umm OK.	
JG	So, and it's always the requirements we end up with because you end up with plain text.	
MN	Umm. Umm.	
JG	Everything you know, we all speak English, but it's all very different dialects. So, you know, there is always that misinterpretation.	
MN	Yeah. Yeah, yeah, yeah, yeah.	
JG	So, this, like the model storming all the way down like we are probably gonna be using this or something close to it in our next production project.	
MN	OK. Hmm.	
JG	Yeah, hopefully it’s a good idea if it's helping hindering the defect identification in the next year. But my instinct is it probably will improve the quality of requirements, but we will find out.	
MN	OK. OK. Umm. OK. Alright. I think we have discussed most of the things that I prepared and yeah, unless if you have something that is still maybe not clear. But I would like to have the chance to come back to you for any clarification based on our discussion.	
JG	OK. Yeah. Uh. Yeah, like if you need to please reach out if we can help in any other way.	
MN	OK.	
JG	Yeah, yeah. If you got time with Hemant, maybe he is a different history to me. Yeah.	
MN	Mm-hmm.	
JG	So, he may have good insights. He has seen a lot of big production projects in large, larger firms. He would have some good insights on this. So maybe reach out to H and see if he can and join you.	
MN	Who?	
JG	HJ is at, I think I see him on the invite for this, but I forwarded it to him but.	
MN	OK, I think I just saw only your email. But thanks for being open to say I can come back to you and have some clarification on our discussion, and I would also ask for your further involvement in the project.	
JG	No problem.	

MN	Is it OK if I can reach out for his involvement?	
JG	Oh yeah, no problem. Yeah.	
MN	OK, alright. Alright, so I will be in touch. I think you got my email address, yeah.	
JG	Uh, yeah, I do. It's misheck.nyirenda@dkit.ie . Yeah.	
MN	Yeah. Yeah.	
JG	Sorry, am I pronouncing your name, OK?	
MN	Yeah. Yeah. Yeah, yeah, yeah, that's correct. Hmm.	
JG	OK. Yeah.	
MN	That's correct.	
JG	Yeah. So yeah, just feel free to reach out if we can help in any way.	
MN	OK. So, I will prepare a transcription for this conversation, and then I'll share with you for any clarification. Yeah.	
JG	Yeah. Perfect. Yeah. Thank you.	
MN	Alright, OK. Yeah. Thanks so much JG for your time. I really appreciate and the insights you have shared are really crucial to my research.	
JG	That no problem.	
MN	Thanks so much.	
JG	I was glad to be found.	
MN	Yeah.	
JG	OK, you have a good day.	
MN	Umm, OK. Thanks. See you too.	
JG	Thanks. All the best.	
MN	OK. Thanks so much.	

E.2 Interview 2 - Part 1

Position: Firmware Lead and Senior Embedded Software Engineer

Experience in MDS: over 7 years

Interview date: 4 June 2024 9:30 to 10:00 am

Speaker	Transcript	Notes
MN	Thank you for accepting to participate in this discussion and I really appreciate for the insights you provide in my research. It is very important.	
DM	Yeah. No problem. I am happy to help.	
MN	Yeah. So, my name is Misheck Nyirenda. I am a researcher in the Regulated Software Research Center here at Dundalk Institute of Technology. And my research is about application of agile practices to address defects in medical device software development. So, before you can also	

	maybe give your introduction, can have your permission to record our conversation?	
DM	Yeah. Yeah. Absolutely. Go for it.	
MN	Ok. Thank you so much, DM.	
DM	Yeah. So, my name is DM. So, my background is, I have worked in Intel for some time. I've worked in Jaguar Land Rover and in the Shannon site. I've worked in Becton Dickinson (BD), which is another medical device company and I'm in Boston Scientific now two and half years.	
MN	Ok.	
DM	So, I have maybe six or seven years in medical device experience, but we have worked in Agile, and I worked in agile in Intel, you know, and Jaguar Land Rover as well you know, so, we have we have experience on both.	
MN	Umm. OK. Oh, that's a vast amount of experience there. Yeah.	
DM	Yeah, I should probably call it like I kind of work as a Firmware Lead currently, so a firmware is kind of like an embedded software really and firmware kind of interchangeable.	
MN	Umm.	
DM	I work as the Firmware Lead for a medical device in Camel. Its multi-site at the moment and also, we use agile and well we use the parts of agile that works for us if that makes sense.	
MN	OK. Mm-hmm. Well, OK. So, that’s like you are hybrid sort of.	
DM	Yeah. So, we don't to be honest Misheck we don't follow it fully.	
MN	Hmm.	
DM	We take the bits that work for us and so use like we do Sprint planning, we do, we follow Sprint, we do the daily stand ups and all that kind of stuff.	
MN	Ok. Ok. Ok.	
DM	We don't do the, I suppose it's due to the phase of the project.	
MN	Umm.	
DM	The way we work is we have a big backlog. So, all our tasks are in a backlog. So, the way it works is we have the requirements for the firmware and the software, you know, for that subsystem. And those requirements are in Jira and they're in, you know, epics and stories and tasks and all that kind of stuff. And then what we do is we define, we assign,	

	I assign blocks of work to developers. So, I say, right, you are on this, this bucket is your bucket. So, for Sprint planning and that kind of thing, what we do is I try and give people the freedom to do Sprint planning on their own, if that makes sense.	
MN	Ok. Hmm.	
DM	So, we set up a 30 minute because we were quite a big team across software and firmware. That was probably 15 to 16 developers on the team.	
MN	Oh OK.	
DM	So, instead of us going through what are you working on for the next three weeks? Because we do three weeks Sprint, you know?	
MN	Umm.	
DM	And you know and go through task by task we have enough trust in the team at this point that the team is mature enough that they can take on like we use story points and story point is kind of a day.	
MN	Mm-hmm.	
DM	So, we do have story points as well. So, that's kind of what we do and we kind of say that developers can work. In my experience, we don't allow for 40 hours like you know we don't assign 40 hours worth of work per week, if that makes sense.	
MN	Mm-hmm.	
DM	You know, because normally a good metric is about 26 hours per week is actually development work.	
MN	Yeah. OK. Hmm.	
DM	The other 14 hours is kind of meetings and all that kind of unscheduled stuff that comes up, that kind of the thing, so that’s the way we work.	
MN	Umm. OK.	
DM	So, we normally say to for a three week Sprint, maybe people take on 10, 11, maybe 12 story points, something along the lines of that depending on how they feel for their three week Sprint and people do that.	
MN	Hmm.	
DM	You know people are, you know, we we're a little bit more mature in our process and we're improving as we go like so.	
MN	Umm.	

DM	So, our performance as a Sprint team is getting better as we go, and you know the more the people get used to it.	
MN	Umm.	
DM	And I find slippages do come in terms of like story point estimates or people create bugs and stuff like that, and they forget to put in like, story point estimates and stuff like that, which makes it kind of, you know, our project management team, then are kind of asking us to make sure that everything is estimated, but that's kind of how we work really so and at the end of every three weeks, then we do three week Sprint, then we do a release.	
MN	Ok. Ok.	
DM	So, we do a system level release across all the components. We have 3 components in our system. So, they all get into our test team then.	
MN	OK. Yeah, I think you are the right people that I can really reach out to in terms of my Research.	
DM	Yes.	
MN	Yeah.	
DM	Good.	
MN	Umm yeah. So, uh, maybe as we proceed with our discussion, would you want me to give a brief explanation of the framework that I have developed, or you are OK for me to ask you some questions that I have prepared?	
DM	Yeah. You can just start asking questions. Whatever makes more sense and most efficient use of the time. I am happy with that.	
MN	OK. OK. Alright. Yeah. So uh, the proposal is that at least you provide views on the agile practices that have been included in the framework.	
DM	Mm-hmm.	
MN	Yeah. In terms of addressing defects, yeah.	
DM	Mm-hmm.	
MN	So, let me just share my screen.	
DM	Yeah	
MN	You can see my screen, yeah.	
DM	Yeah, I can see it.	
MN	OK. So, I would like us to start with the requirements gathering stage.	
DM	Mm-hmm.	

MN	So, what would be your thoughts on using user story in requirements gathering stage to prevent defects?	
DM	So, we are talking about the requirements phase, looking at from the users perspective and how does that help us kind of avoid defects, is it?	
MN	Yeah, yeah, yeah.	
DM	I mean, it definitely helps to view things from the user's perspective. The only thing is, you know, in medical devices the requirements that I've seen they do use user stories like there is kind of you know, this is the user phase, and this is what the customer or the end user needs.	
MN	Hmm. Mm-hmm.	
DM	So, the treatment, like we don't do user stories at the requirements phase to be honest with you. So that would be a change for us to be honest with you and the requirements are very much traditional and as you know, I presume you are aware, like I know this is very V-Model and that's very much the way you know, that's exactly how we do things.	
MN	OK, OK. Mm-hmm. Umm. Mm-hmm.	
DM	And so, I mean, it would be a shift.	
MN	Ok. Ok.	
DM	I mean, I'm not saying it's something that you know it's a bad thing or a good thing or anything like that. I'm just saying it's not currently how I have observed it been done in medical devices, so this would be a change.	
MN	Mm-hmm. Ok	
DM	So, if we went down to how, you know, and I'm familiar with user stories in terms of the requirements phase from software products and websites and stuff like that.	
MN	Hmm. Mm-hmm.	
DM	But to actually bring this to a medical device, and that would be new, it would definitely be new. So, it would be something that, to be honest, should be trialled I would imagine. It would be good to see a flow like, you know, get a medical device for yourself and think of a medical device, you know, something that's in the field, something that you can readily get access to and then kind of fit it in here just to see how it works, if that makes sense. Does that make sense to see how? That's what I'd like to see here, to be honest.	
MN	OK, alright. Umm. Hmm.	
DM	Use case and again like use case onsite testing it's difficult again at the requirements stage for a medical device because we don't have anything to test until we get.	

MN	And. Mm-hmm.	
DM	So, what tends to happen is, there's different phases in medical devices, sort of, you know, you have to kind of prove your technology, which is the first point. So, normally what happens is you prove your concept, and you say right.	
MN	Mm-hmm.	
DM	And you do that using a kind of maybe off-the-shelf part or something like that. So, you buy a microcontroller, and you buy a GUI, you know, you get a GUI, and you get some software, and you prove it. Prove it at a system level that the technology works, that you can drive a pump to push fluid around a patient.	
MN	Hmm.	
DM	Or you can, you know, you can do that kind of thing and then you move to that kind of then planning how to do that at scale and you say right now what parts would we need to use and so it kind of advances in that way.	
MN	Umm. Hmm.	
DM	So, it's kind of a little bit different.	
MN	OK. Mm-hmm.	
DM	And so, like that medical device and I know this is probably more geared to a kind of a software product and I'm just, I'm kind of, yeah, yeah, I know.	
MN	Yeah, yeah, yeah.	
DM	And to be honest, I understand that.	
MN	Mm-hmm.	
DM	So, what do you think the target is here M? Is this like, are we saying this is something like a website for, you know, for a hospital or I am just trying to figure out what are we thinking here is, you know, what are we targeting here?	
MN	Yes. So, uh, the target is any kind of software that is a medical device.	
DM	Is any software that's in medical device?	
MN	Yeah.	
DM	OK, OK.	
MN	And any software that is used for health related. So, to make sure that for example, even the records management if you have poor records within your system, defects may creep in, and there was a case where a wrong patient was linked to a	

	different medical record and the treatment that went on resulted in the death of the patient.	
DM	OK.	
MN	Yeah. So, any software that is used for medical purposes.	
DM	Ok.	
MN	Yeah. So, to make sure that that software is developed without, at least without defects, yeah.	
DM	OK so. So, what we are saying is, you know, this is the existing V-Model requirements gathering and what we are saying is how the blue on the outside here is how that fits in this, is it?	
MN	Mm-hmm. Yeah.	
DM	This is OK. OK.	
MN	Yeah. How would these practices help in addressing defects? So, in the left side I am looking at prevention because these are early stages.	
DM	Umm.	
MN	So, as much as possible, we do the right thing as early as possible so that we prevent defects and then on the left side I am looking at if any defect crops up in those stages, we need to identify that using agile practices and then correct the defect, yeah.	
DM	OK. OK. So, I think this is just my feedback, is like it's probably too early then to bring in customer. That kind of a thing on the left hand side.	
MN	Mm-hmm. OK.	
DM	So, normally because like we don't have anything to test until much further down if that makes sense.	
MN	Mm-hmm. Ok.	
DM	So, we are over on the right hand side, really. And you know it's coming up that side where we are going to be actually, you know, have something to go to a customer.	
MN	Mm-hmm.	
DM	I mean, we can definitely go to customers with requirements and say, you know, if we had this product in principle could we actually use you know, could we use that?	
MN	OK. Mm-hmm.	
DM	But we would have to go through the FDA, and we would have to get something that was ready to test.	
MN	Umm.	

DM	So, we would have to get approval for a clinical trial really for a medical device and that's before we can kind of use it on customers and that kind of thing.	
MN	OK, OK. Mm-hmm.	
DM	So, it's much too early to bring customers in for a physical medical device up on the requirements stage if that makes sense.	
MN	OK. OK. Hmm.	
DM	And now again we can get customer feedback and say if we had a medical device that fixed this problem, would that be useful? What kind of things would you like to see from it? So, it's not that we can't get customer input, but from the device perspective it will be much, much more over the further side where we can do clinical trials where you know that, that kind of.	
MN	Mm-hmm. OK.	
DM	So, what I would advise you, my advice would be to kind of take the medical device.	
MN	Mm-hmm.	
DM	So, have you have you accessed to, I presume you have in the college IEC 62304, the medical device software standard. So, I would definitely try and bring that in here because everything we do in medical has to conform to that for software.	
MN	Yeah, yeah. And. Mm-hmm. Yeah, yeah.	
DM	So, that has to be kind of, you know, we have to kind of, you know figure out that as well if that makes sense.	
MN	Yeah. OK so. I am specifically looking at the software part, whether it will be used within the medical device or a stand-alone, say when we are developing that software it has to uh, we need to make sure that we are doing the right things starting early so that defects can be prevented.	
DM	Mm-hmm.	
MN	So, the idea is if we can use agile practices as included at the stages.	
DM	Yeah, yeah, and to be honest, I definitely it's a good idea and I definitely think we can. I definitely. I definitely think we can.	
MN	Mm-hmm.	
DM	My feedback is just, you know, a medical device is very, very different thing to a patient management system on a website for you know, for a hospital that would manage, you	

	know, patients, files and records and stuff like that an actual medical device has to go through something very, very different.	
MN	Hmm. Umm.	
DM	And the software on a medical device also has a very, very different workflow. So, you just have to kind of like everything we do is kind of based on, you know, the next milestone. So, we've to do IEC 60601 compliance testing now on our product soon. So that you know. So, there may be changes on that and you know.	
MN	Umm. Ok.	
DM	So, there is a very clear path we have to follow because we have to prove that to our quality system and we have to prove that we have met, we have done everything between IEC 62304, and we have met all our standards before we can actually get the medical device into the field.	
MN	Hmm. Umm. Hmm.	
DM	So, I would suggest that this is a good system along with what we do if that makes sense, along with the existing path you know and just kind of keep that in mind if that makes sense.	
MN	Umm. Umm. Hmm. OK.	
DM	So, and sorry, as I said, I do think that requirements gathering user stories definitely help.	
MN	Umm.	
DM	Yeah, definitely help.	
MN	Mm-hmm.	
DM	I mean, we know the benefits of this stuff in normal workflow in, you know the Netflix website or the Google website.	
MN	Ok.	
DM	You know, we know this stuff works.	
MN	Hmm.	
DM	We know this stuff adds quality like and I am particularly looking down at like we do a lot of this stuff on the right hand side. We do already so, so now.	
MN	Hmm. OK.	
DM	But so, we do like unit testing. We do integration testing. We use code reviews. We do continuous integration, CI/CD. We do refactor so everything you see up on the right hand side in blue we do and that to be honest with you, that is	

	massive. That is absolutely a huge driver of quality in our project and so that's very, very valuable.	
MN	Hmm, OK.	
DM	And there's just like, there's no, it's very hard to quantify how much, you know, code reviews, just take code reviews like people are catching my mistakes and that kind of the thing. And then it's like, oh, that's great they are caught. I thought something worked right. I read the code in a certain way, and I wrote it in a certain way, so I automatically assumed that it works, whereas a code reviewer comes on and the code reviewer says, well, that doesn't make sense because of this module isn't going to talk to this module in the way you think it's gonna talk to that module. So, you know, can you explain it to me?	
MN	Umm. Hmm. OK. Mm-hmm.	
DM	And then you go, oh, I made a mistake. So, I have to go back and fix it. So, code reviews and the other thing I would say is massive is CI/CD.	
MN	Hmm.	
DM	So, when you see continuous integration, that's huge because what a very mature pipeline for software firm or code development in medical or anywhere looks like is you know and you would write your code you do it on your editor and then you push software through a pipeline to a build pipeline that runs your unit tests, your static analysis, your integration tests, all automated in the cloud pipeline and then that fails if you made a mistake. So, if maybe my unit test fail now after a change I made, that lets the developer know instantly they have made a mistake. I have to go back and fix it and you get really, really, really good quality code out the other site to be honest.	
MN	Yeah. OK. Hmm.	
DM	So that's huge. So that, and to be honest, this is new for medical devices because I come from an automotive background. So, I am trying to bring that into what we are doing at the moment to add quality.	
MN	Mm-hmm. Ok	
DM	So, in like automotive, this stuff is kind of second edge in automotive, CI/CD. All that kind of stuff. Whereas in medical devices and medical things are slower to change, it's like well, we just need to pass this project, don't we just need to do this, we just need to do this depending on our	

	project, which is another thing, and you have to consider for medical devices is the safety classification.	
MN	Hmm. Umm. Hmm. Umm. Yeah.	
DM	Have you heard about the safety classification class A, B and C medical devices.	
MN	Yeah, yeah, yeah, yeah.	
DM	Yeah. So, you have to take those into consideration as well. So, Class A is a or an IVD which is a diagnostics device basically.	
MN	Mm-hmm. Yeah, yeah.	
DM	So, if it's one of those in one of those classes, you don't have to do as much like on the software unit testing on the integration testing and all that kind of stuff.	
MN	Hmm. Mm-hmm.	
DM	Whereas if you get to Class B, you have to do a different level of integration and unit testing and then you get to Class C which is a life sustaining or critical. OK, so you could kill someone if something goes wrong. We have to do extra. You know your integration test and unit testing. We have to cover all of those. Tick all of those boxes. So, that also moulds how a software organization responds to V-Model as is there because in Class A you don't have to do any unit testing.	
MN	OK, OK. Hmm.	
DM	If you have a Class A device, so that takes that whole circle out and it's the same in integration testing because it's just Class A and then you just do some system testing that it works as you know as it's supposed to kind of a thing and that's fine and you know so you can skip some of the stages in the pipeline. And so, it's something just to consider as well.	
MN	OK. Hmm.	
DM	But going back to the left hand side, I think on the left hand side just be careful with the like, you know, user stories and customer and onsite customer meetings and visits. Definitely good to get customer feedback early. But you don't actually have something to test over there. You don't have something to show you don't have, you know it's too early for medical to actually over on that side to kind of go out to a customer and you know, because basically what some of these things suggest is that you know, we are ready for kind of clinical trial kind of a thing and you are not, you know, if you are still in your requirements analysis, you just don't have anything to test. So, it's kind of more about	

	customer feedback, customer input from a medical. I would like just to call that one out.	
MN	OK. Mm-hmm. Hmm.	
DM	And so, I again, integration testing, we don't have anything to integrate you know to run integration testing on the left hand side if that makes sense.	
MN	Hmm. Hmm.	
DM	It's not until we actually get come up the other side where we get, you know, our implementation done on the bottom. Once we get the implementation done, then we start bringing in our unit testing, our integration testing and then that, you know, it's on that site to be honest.	
MN	Hmm. OK, alright.	
DM	Does that make sense?	
MN	Yeah. So, yeah, you have clearly explained the current practice, but now looking at the proposal to say if we followed, we applied these practices.	
DM	Mm-hmm. Mm-hmm.	
MN	Uh, for specifically looking at the software part would these practices help. For example, at the different stages to prevent defects.	
DM	Yeah, I mean they have to, right? I mean like if you, you know, if you're in your requirements gathering phase, if you're taking your customers, you know, what does the customer want from this?	
MN	Mm-hmm.	
DM	What are his or her needs from this device or software? That you can only know if you get that right, that feeds everything that comes down, you know at the other side and you don't find yourself at the end coming out to a customer site and saying it would be great that medical device does this, this, this and this. While if you just made one other change, you could have dealt with all these other problems and fix. You would have a much better product if you just did that. So, definitely it does help, and customer feedback early is huge.	
MN	Mm-hmm. Yeah. Hmm. Hmm.	
DM	Model storming and pair programming, like test-driven development is massive as well.	
MN	Hmm. Hmm.	
DM	Like medical, like test-driven development would be very, very useful and medical devices you know designed for test I mean that feeds into the CI/CD uh the continuous	

	integration, continuous development all that kind of stuff and like I said test-driven development then means as you are starting your architectural design, your detail design, your unit implementation, you know, you have good quality tests that are ready to kind of do. So, that definitely helps.	
MN	Hmm. Mm-hmm.	
DM	And I think that would help, yeah.	
MN	OK, OK. Right. So, I would have loved if you discussed the practice at each stage each practice at each stage. So, that would maybe clearly give me picture to say how I can change or what should I improve within the specific stage in regard to agile practices. So, of course you have touched on a number of them already, which is great, yeah.	
DM	Mm-hmm.	
MN	Yes. So, relating to the standards you mentioned earlier on.	
DM	Umm.	
MN	What would be your thoughts in Implementing this framework in relation to the standards? For example, the IEC 62304.	
DM	Yeah, I mean do you have, I don't have a copy of IEC62304 close to hand because what happens Misheck is companies like Boston and BD, what they do is they take, we have our own quality department. What happens is we adjust like we take 62304, we translate that into Boston specific process. Now it's all very similar to what we have to show requirements we can sign off.	
MN	Mm-hmm. Umm. Ok.	
DM	We have requirements analysis, so we do everything you see in the V-Model, but in a Boston specific process. So, what happens then is our quality team then go in and say OK and we are happy with that. We will sign off on it. We sign off on those documents. They will all go into the we all upload them. Once they are done and effective and signed off, they go up to the cloud, they get stored, they get effective, they get signed off and then we ship them to the FDA for analysis on their side.	
MN	Mm-hmm. Ok. Ok.	
DM	And so like, we have our own internal version of 62304.	
MN	Umm.	
DM	So, my advice to you would be just to kind of consider that as well, just to consider that you know, we conform to, every company in medical confirms to the 62304, but every	

	company has their own internal way of doing it if that makes sense.	
MN	OK. Mm-hmm. OK. Umm.	
DM	So, it's kind of, we just have to tick the boxes for it.	
MN	Hmm. OK.	
DM	But everything you see here; we do everything that you are seeing here in the grey circles.	
MN	OK. OK. Umm. OK.	
DM	We do everything here, so it's quite similar to that, to be honest.	
MN	Mm-hmm.	
DM	So, like we would have to be honest, we would have like, you know, software detailed designs, we would have software architecture design documents, we have, you know, test plans, V&V plans, verification, validation plans, you know all of that stuff.	
MN	OK, alright. Hmm.	
DM	So, it's all. It's all kind of, yeah, it's all.	
MN	Yeah. OK.	
DM	Misheck, I have to go because I have that call but we will catch up on Thursday.	
MN	OK, OK. Yeah. Yeah.	
DM	OK, talk soon.	
MN	OK. Thanks. Thanks so much, David.	
DM	No problem. Bye.	
MN	Yeah. OK, bye.	

E.2 Interview 2 - Part 2

Position: Firmware Lead and Senior Embedded Software Engineer

Experience in MDS: Over 7 years

Interview date: 7 July 2024 9:30 to 10:15 am

Speaker	Transcript	Notes
MN	I just appreciate the time you set aside to attend to my research. I really appreciate.	

DM	Yeah, no problem for that.	
MN	I would also ask you to allow me to record our discussion.	
DM	Yeah, yeah go for it.	
MN	Ok. Yeah, so in our last discussion meeting I think you shared more information about how you work and some of the practices that you use. So, I would still want us to at least go through the stages and looking at those practices and may be specifically pointing out your views on how you see those practices being useful in addressing defects in medical device software development, yeah. So, let me just share the framework again. I hope you can see it, yeah?	
DM	Yeah. I can see it.	
MN	Ok. Yeah. So, starting with software requirements gathering stage there. You can share your views on the practices included there and then looking also at the complementary use of those practices like using user story and onsite customer together in requirements gathering and use case and onsite customer together in requirements gathering and also user story and use case together in requirements gathering in terms of preventing defects at that stage.	
DM	Yeah. I mean, yeah, I definitely think there is a massive, there is lots we can leverage there, Misheck. At the requirements gathering phase like so as you say, if you take you know, your user stories or your you know your customer visits and I mean for us it would be you know visiting physicians. And maybe you know from a Gen 1 to a Gen 2 device like there might have been defects in Gen one and you go out to a customer and one of our customers and we say listen we are in the life and the first Gen of a project and we are gonna hardware upgrade and all that kind of stuff and then you can go through the customer complaints and then you can go through the defects at that point with the customer and the customer can say we have defect with this. You know, you are actually getting feedback at the requirements stage from the customer is a massive help yeah for sure.	
MN	Mm-hmm. OK. Uh so, in terms of complementary use of these.	
DM	Yeah. I mean, I think the onsite, the strong because as I see it like the customer feedback at the requirements stage is vital really to addressing defects because sometimes, like were engineers and so we think about how something will work in an engineer capacity, right. But a doctor or a physician or a nurse, or someone in the medical setting, has a different perspective because they can say, well, your cable is 5 meters and it's long, so that's not going to work for us in the setting of an operating room or the setting of a clinic or that kind of stuff.	
MN	Hmm.	

DM	So, getting that outside feedback at the requirements stage is great because then you can say look you are not getting to the end and you are saying our cable for our pump is 5 meters and the customer are saying well, we don't, you know we can't choose that.	
MN	Umm.	
DM	It's a great idea in theory, but in practice it's not, so it definitely does address the customer input at the start.	
MN	OK.	
DM	Definitely helps. You know, we need that input.	
MN	Mm-hmm. OK. Do you have any other practice that you suggested to use at this stage?	
DM	I mean you again user stories are.	
MN	Sorry I lost you there. I think I lost you there. I don't know where this my network or.	
DM	I think it could be my network. Can you hear me?	
MN	Yeah, I can hear you now.	
DM	Yeah, I think it could be my network Misheck. I'm having some issues with my broadband at the moment.	
MN	Oh OK.	
DM	And but yes, I definitely think use case models and user stories are definitely a vital way of you know, gathering requirements and that potentially may not be done currently in medical devices.	
MN	Mm-hmm. OK, moving to the software requirements. What would be your comments on model storming, behaviour-driven development, pair programming and onsite customer or proxy in the requirements analysis stage.	
DM	Yeah. So, you mean at the requirements an analysis. I mean look, at that point, you know, we have to try as the software team at that point has to translate the requirements and see are these feasible, right. So, what you're looking at here in medical usually is like a phase in a medical device called EFS, which is Early Feasibility Study.	
MN	Mm-hmm.	
DM	So, at that point you could be looking at a proof of concept project you know. So, it's not quite, I mean it could be not quite pair programming or test-driven development at that point. Uh that's onsite customer proxy, I mean Integration. Yeah, I'm just looking on the left hand side. I mean, we tend to follow, I mean Misheck you are familiar with like you know this inside diagram here. Where did this come from?	

MN	Yes. So, this is the V-Model.	
DM	Umm. Yeah. So, what we thought, I suppose this is something. This is the V-Model but.	
MN	Yeah. So, what I have done here is to bring these agile and the V-Model together, yeah.	
DM	Hmm. So, what I would like to see, to be honest with you is because some of this what I would like to see is, I mean, I have, let me see I have a diagram of the V-Model for IEC.	
MN	Mm-hmm.	
DM	OK, so, you know last week we spoke about IEC and 62304.	
MN	Yeah, yeah, yeah.	
DM	So, let me share this one in the chat, right. And this is probably more applicable to a medical device.	
MN	OK. Oh OK.	
DM	And because like to be honest with you, it's similar, it is quite similar, to be honest.	
MN	Umm. Mm-hmm.	
DM	It's just, it's different. I can't for some reason I can't access the chat.	
MN	Ok.	
DM	One second. Just give me a second. Umm. Can I share my screen for a second?	
MN	Yeah	
DM	Let me share the screen, so let me know when you can see that.	
MN	OK. May be I need to stop sharing.	
DM	Maybe I might turn off my camera Misheck because I know the broadband	
MN	Yeah, I know. OK, OK.	
DM	Let me just turn off that.	
MN	But I can see the screen now yeah.	
DM	You can see it?	
MN	Yeah.	
DM	Perfect. So, now if you look at that, that's more of what, what our you know what our development looks like in terms of the V-Model, right.	
MN	Umm OK.	
DM	So, our V-Model looks more like this, right. So, you have your system requirement at the start. Your software requirement,	

	your software design process, then your coding is at the bottom, right. So, you are coding, but your unit and integration testing then is on the other side of the V. So, when I look at your model, I see it on the left hand side coming down, whereas our unit and integration testing phase is on the other side coming up.	
MN	Umm.	
DM	So, it's more a way of validating the design process, if that makes sense.	
MN	Hmm.	
DM	So, I would like to see the agile stuff targeted here because everything in medical like so everything in medical devices is 62304.	
MN	Umm.	
DM	So, like for this, for your work to be applied to medical devices, you will have to bring 62304 at every phase of what you do. Because like I was saying the last time in Boston, we what we don't follow like there is a standard you can take off the shelf we don't, but we follow our process that follows this.	
MN	Yeah. Mm-hmm.	
DM	So, our software quality department takes 62304 and they develop trainings and processes for us as engineers to follow.	
MN	OK.	
DM	But all of this would follow this model, so everything would come back to this so quality management system configuration, change management, risk management, safety processes and then the planning process.	
MN	Mm-hmm. OK. Umm.	
DM	So, it's very, very this is the best model for medical devices.	
MN	Hmm.	
DM	If you look at your model, it looks different to this, if that makes sense. That so that that that would be my only concern. So, this is the process that we follow that makes sense.	
MN	Mm-hmm. OK, OK.	
DM	So, it's kind of hard for me to talk to the other process too much now like is I can talk to parts of it, but it's like if you wanna start, if you want to share your screen again.	
MN	Mm-hmm. Ok. OK, alright.	
DM	I can't go to chat.	
MN	Yeah. Can you see it?	
DM	Yeah, I can see it.	

MN	OK.	
DM	So, do you see on your model that your integration testing here is actually on the right side actually. So, if you look at it is on the right side.	
MN	Yeah.	
DM	The only difference is on the left hand side here where your blue boxes are coming in, where you're trying to pull in agile. You have integration testing listed there as part of the architectural design, and that's much too early for integration testing because you are not kind of what are we integrating at that point if that makes sense.	
MN	Uh, you are developing the architecture, but you need to have integration testers there to check whether things will fit together properly. So, that's why we are including integration testing. It's just because in agile, uh, the terms or the names of the practices are like that.	
DM	Mm-hmm.	
MN	But what we are saying there is that you need to involve integration testers so that they should be able to write their tests based on what their architecture there and they should be able to figure out to say, OK, will these things fit together, and they can point out some of the issues that would crop up later on.	
DM	Ok. Ok.	
MN	Yeah, yeah.	
DM	Yeah. So, I would just specify that, and I would make that clear in the document that it's integration test team involvement as opposed to integration testing because when I see that, I think you know we are actually doing our integration tests.	
MN	Yeah, yeah.	
DM	So, integration test planning, I would put in there and that makes sense.	
MN	Yeah, yeah. Umm.	
DM	That does make perfect sense, and like pulling in the integration test team at that point, to look at the architectural design, even if they don't point out problems, but they can, like, cline their testing then as well, so they can say, right we need to have components. These are the tools I need to get. I need to validate these tests.	
MN	Yeah.	
DM	So, the power here is from a medical device perspective, the power here is the integration test team then can start planning what tooling they need because all our tooling in medical devices needs to be validated as part of the process as well.	

MN	Umm. Umm.	
DM	So, in a lot of cases, we can't just say, right, I want to use GCC, I wanna use CXX test. These are the tools I want to use. We have to, as part of our submissions, we have to provide validation reports for our test tools.	
MN	Umm. Mm-hmm.	
DM	So, that definitely would help on the that would like not just as prevent the bugs like not just in terms of just how to improve the software development life cycle. I think that would be a that would be a big help there.	
MN	OK. Alright, so uh, what about test driven development?	
DM	Again, at that point at the requirements analysis like above it as well, I think I mean like if you go up to the second circle software requirements analysis like to have test driven development there or pair programming we are not supposed to be programming at that point.	
MN	Hmm. Hmm. OK.	
DM	So, we are supposed to be programming down the bottom. So, once, like, once the architectural design is completed, signed off, then we do our detailed design and then there is other documents there that are not listed.	
MN	Mm-hmm. Hmm.	
DM	So, at the detailed design phase, right, we have to come up with our configuration management plan, which is how we are going to manage the software. What are we going to use? Are we going to use Gates? So, it's too early to see like that. That's kind of what I am saying about the blue circles. It's too early to see pair programming and test-driven development at this point, and it's too early to see integration testing. I wanna see integration test planning at that point.	
MN	Hmm. OK.	
DM	So, what I would say is I would shorten that to, not. I would say instead of integration testing in those blue boxes, I would just say IT/UT planning, that's what I would say.	
MN	Hmm. Ok.	
DM	And just to call it out new say right, I am doing my integration testing, and my unit test plan are gonna start at this point to the process, and I think that's very useful.	
MN	Mm-hmm. Ok. Alright. Yeah, that's in a good suggestion there. That's a good view. Yeah.	
DM	And then when we get come down to the architectural design, again having that I mean at that point like I am not sure customer	

	input like we wouldn't be going back to the customer at that point.	
MN	Mm-hmm.	
DM	So, the same with the detailed design.	
MN	Ok.	
DM	So, I think like once we get our requirements, I think requirements gathering is the best place to bring the customer in.	
MN	Umm.	
DM	Requirements analysis then is when you bring your engineering teams in and say right are they are these requirements possible in the timeline in the budget? Are they technically possible? So, can we actually give the customer what they're like? You know, if the user is asking for something like, you know that's not medically possible at that point, can we deliver it because maybe we can't maybe, maybe it's just not feasible.	
MN	Mm-hmm.	
DM	So, that's why we do the feasibility study. And so, then as you are coming down to the detailed design, like, again you can have the integration test team planning there and then the like. And then we are down in the implementation and that's the interesting part, to be honest with you. That's where pair programming, test-driven development is really, really test driven development particularly I think is a way that companies can develop quicker and develop and more reliable and robust products. To be honest with you, that's my opinion, I think.	
MN	OK. OK. Umm.	
DM	So, if you look at like and that would be helpful. So test-driven development would be helped by having integration test planning and unit test planning at the earlier phases and the architectural design and all that kind of stuff that makes your test driven development much easier when we actually start implementing our software.	
MN	Mm-hmm. OK. So, you are suggesting unit test planning at which stage?	
DM	I am suggesting at the architectural design and the detailed design I would have unit test and integration test planning at those phases and then I would like and then that if you do use test driven development in for your actual implementation you have a very good head start on that stuff.	
MN	OK. OK. Umm. OK.	
DM	And I think test driven development fits really, really well into CI/CD. So, I would where you have CI continuous integration,	

	I would make that CI slash CD so it's continuous development as well.	
MN	Mm-hmm. OK.	
DM	So, that's you're continuing to integrate your continuing to develop and in like what I would also point out there Misheck is kind of I would look for kind of point out for like CI/CD pipeline and best practices.	
MN	OK. Hmm.	
DM	So, static analysis and all of that stuff at that point in the implementation stage. So, your software security analysis, static analysis, unit test, integration testing all in a build pipeline that kicks off each time the developer, you know there's a nightly build or there's a daily build or whatever it is, you know that kind of stuff.	
MN	Mm-hmm. Hmm.	
DM	Those are really powerful parts, to be honest with you, that are really the parts that really make the difference.	
MN	OK, alright. OK.	
DM	So, I would definitely expand that CI/CD part to me is and yeah, but the unit implementation is and then I would call out kind of what are the best like best practices like I can't share, just give me a second.	
MN	Hmm.	
DM	My Internet is definitely under blink today. Can you hear me, Misheck?	
MN	Yeah, but I think now it's better, yeah.	
DM	OK, OK. And but so yeah. So, I definitely I can't access the chat, but I would look at and I know it's recorded anywhere.	
MN	OK.	
DM	So, for CI I would make that CI/CD and look up like best practices for CI/CD pipelines and the different stages of a good CI/CD pipeline and add some of those in there.	
MN	Mm-hmm. OK.	
DM	Because you are telling people and look this what we want. This is how we enforce best practices, so we do our coding standards, we have automated style guides, we have all of this stuff and Misra checks all that kind of stuff for our software development, and that's the power like to be honest with you.	
MN	Hmm. Hmm. OK, alright. And then still going back on the left side there. I think there are some practices where I also want to get your views on like behaviour-driven development.	

DM	Mm-hmm. Yeah.	
MN	At the software requirements analysis there.	
DM	So, just are we talking about just refining the requirements here using BDD?	
MN	Yeah. So here you do, I think it's what you have explained to say you need to have like programmers, testers and architects together looking at the requirement. So, after that process then each one could now proceed like testers would start writing test cases and the architects would be looking at how they would come up with the architecture of the software and the like.	
DM	Yeah, I mean that's possible. Again, it's probably too early for test cases. I mean, it's OK probably for system level test cases, so that's another type of testing as well.	
MN	Hmm.	
DM	So, thought like at the system level and but in terms of the actual unit and integration testing, I mean it's potentially possible that you would have some system integration level tests. But there's also, like software integration tests that you wouldn't be able to plan at that stage.	
MN	Ok.	
DM	And because of different modules, different components in your systems, you are still going through your requirements analysis here and you are still figuring out what's feasible, what it's the hardware I need, what's the software I need**,** you know. So, you are still in that process and in medical devices, you know, the software is very, very, very much driven by the hardware. So, we are figuring out what are the components, right? One of the massive challenges we have in here is, what component? We know the ideal component we want, but what component can we actually get and what component can we guarantee is going to be, we are gonna be able to get for the next 20 years, you know, that kind of way.	
MN	Mm-hmm. OK. Yeah.	
DM	So, there's lots of stuff in there that's driven purely by hardware decisions. So, all of that stuff that has to be considered there in the software requirements analysis stage.	
MN	Mm-hmm. OK. And then model storming.	
DM	So, what are we saying by model storming? What do we mean?	
MN	OK. So, this is where you can for example when you are conducting behaviour-driven development, uh, just for clarity in terms of a requirement you would come up with maybe a diagram or a sketch to say this is something that we mean. So,	

	other people would easily understand the requirement through that.	
DM	Yeah, that would be useful. I suppose at the architectural design stage.	
MN	OK. At the architectural design?	
DM	Yeah, I think that adds value at the design phase.	
MN	OK. And then simple design.	
DM	Probably not really to be honest with you because in medical if like if we are delivering a class C, life is sustaining medical device. We have to kind of be very careful with the phraseology we use, but we also have to, you know, we have to do a detailed design, to be honest with you. You know, we have to be seen to be going into detail and I know there's system like simple flows and that makes, you know. that can illustrate what we are trying to do quickly and simply but yeah, we just need to be careful about the phraseology we are using there.	
MN	OK. Alright. Then now going to the left side and OK, maybe before we go to the left side, are there any other agile practices that you suggest on software requirements analysis or architectural design or detailed design and unit implementation, the stages that we have discussed that you would want to be shown there?	
DM	One thing I am not seeing Misheck cause are we following like, are we saying that we are gonna use as part of the agile process we are gonna use an agile methodology like Scrum? Or are we saying anything like that?	
MN	No. So, and in this case, we have taken practices from different agile methodologies like Scrum, XP and the included them in this framework.	
DM	Mm-hmm.	
MN	So, the proposal is that, yeah, medical device software developers would use this.	
DM	One thing one thing I'd like to say then right is like Scrum doesn't necessarily work for everyone, right. It doesn't, to be frank, but I find that the best way I have seen it working across three companies is people take the best parts of it, that works for them. And one of the best parts of Scrum is the daily stand up that it's the very, very best part of it.	
MN	Mm-hmm. Hmm, Ok.	
DM	So, I would like to see like I mean and that encompasses everything you are seeing here, to be honest with you. Like that encompasses not just the development team. Like a simple and	

	the best stand ups are like you can if you have a team of 10 people best stand ups are 15 minutes long.	
MN	Mm-hmm. Hmm.	
DM	What I did in the last 24 hours, what I'm looking at two and for the next 24 hours and if I have any blockers and if I have any blockers, I list the I like every everyone's update is 30 to 40 seconds max, and it's caught a 30 to 40 seconds max, and we just roll on. It's just a methodology of sparking discussion around different areas. So, I might be an expert in something that you are seeing a blocker in, so I can say oh Misheck I am familiar with that. Let's catch up after the call, so we're not wasting everyone's time.	
MN	Hmm, OK.	
DM	It's just you know, it's a short, sharp, very precise meeting, but it adds so much value, I think.	
MN	OK. So, looking at in terms of addressing defects. Do you think daily stand up would help to address defects?	
DM	Absolutely. Absolutely.	
MN	OK, alright.	
DM	Yeah, absolutely. Because it anything that increases communication in the team can help defect prevention because I might be the subject matter expert in one area of the software, and you know someone else is on holidays or whatever it is, and I might, or I am on holidays and then someone else might you know be less familiar with it and that kind of thing.	
MN	Mm-hmm. Hmm.	
DM	So, having that open, as I said, anything that increases communication helps Misheck because as a software engineer I have my ideas of how my code should work and I could say oh the requirement says I am working on requirement X requirement X says the system needs to respond within 200 milliseconds and I am having an issue with that and then someone who was actually better placed under requirements says well, D, it actually is 150 milliseconds here. Gonna have to go back to the drawing board on there. So, it's having that open crack across the team. Communication is massive to be honest with you.	
MN	OK. OK. Alright. So, you would suggest this one, at which stage?	
DM	I would suggest at like to be honest, you are definitely at the unit implementation, so definitely at the very bottom and all the way up the right hand side.	
MN	OK.	

DM	And but I can see value for that.	
MN	Ok.	
DM	Yeah, probably that's the best place to bring it in. So, during the implementation phase, I would see daily stand up and I would also like to see work tracking using Jira or something like that.	
MN	Ok	
DM	You know, I don't know if you are familiar with JIRA using like a scrum board on JIRA is very, very powerful as well because it's you let you track tasks.	
MN	Mm-hmm. OK. Umm.	
DM	It lets you monitor tasks. It lets you have a work log of tasks, so and it helps not only project managers estimate and it to be honest with you, you think project managers don't have an impact on defects, right? But they do if their timelines are incorrect.	
MN	Umm.	
DM	So, if they have an incorrect timeline in place and an incorrect you know if their timeline is off by three months, and they don't understand that that becomes a crunch point at some point because they will tell developers to get things done faster and that's when defects get introduced.	
MN	Hmm.	
DM	So, having an accurate and realistic picture of the timeline is massive for defect prevention.	
MN	Umm. Mm-hmm. OK. So, you are suggesting to have JIRA or Scrum board.	
DM	Definitely JIRA, yeah and some method of tracking you know outstanding work left in the project, so we need to know like when we are in the implementation phase, we need to know how long that's going to take and we need to know how long the unit test is gonna to take.	
MN	OK. Hmm. Umm. Mm-hmm.	
DM	Because eventually when we get to acceptance testing and sign off and submissions and all that kind of stuff. Again, like if our project management team think that's gonna happen in June, when we know, we are going to still be in the unit testing phase in June, there can be a massive crunch point then because you know projects are promised and budgets are crunched, and you know so they have massive impact on defects.	
MN	OK. Mm-hmm. OK. What other practices would you suggest?	

DM	Umm to let me think. CI/CD is massive, the daily stand up is massive, Scrum Board is very powerful, and I would also like to see that the configuration management tool that you use links to your JIRA.	
MN	Yeah. Umm. Umm. Mm-hmm. OK.	
DM	So, we develop, a lot of companies use tools that link together. So, we use Bit Bucket and Git and if I push changes to software to Git, I can track that to the Scrum Board on JIRA.	
MN	Mm-hmm.	
DM	So, I can go into JIRA, and I can check this bug has been addressed in this pull request. So, then as a team lead, I can go in and say, well, that doesn't actually address that. You know, I can go in and say Misheck did this piece of firmware, piece of software and he thinks this is going to fix this bug and I have traceability through my tool set to kind of identify that stuff.	
MN	Mm-hmm. Hmm.	
DM	So, I would say like you know, tool set traceability is another really, really powerful tool to address, you know, defect prevention here and defected identification.	
MN	OK. OK. So, you are suggesting which tool?	
DM	There's multiple right, there's multiple tools, but I am just giving you an example.	
MN	OK, OK.	
DM	So, we use Bitbucket and JIRA. So, JIRA is our workboard tracker, so we track our work in JIRA, and we develop using Bitbucket. Our git repositories are stored in Bitbucket. Our work that we track is in JIRA and they are linked together so they work seamlessly together. So, I can tag work in JIRA to Bitbucket and vice versa and we can do our formal pull request using JIRA.	
MN	Ok.	
DM	So, it's very, very powerful to have that traceability in the project and then at the end we can export a work log for the entire project for all the tickets in JIRA and we can attach that to our submission. So, the FDA or whoever the sanctioning body or the sign off body is can have access to a full work log. It's all and all that information is auditable. You can't develop a medical device without that, so we can't come to the FDA and say we want to submit a Class C medical device, and they can say, OK, we want to audit, well, how you developed it.	
MN	Hmm. OK.	

DM	So, they want to see your JIRA. They want to see your Bitbucket. They want to see all that stuff. So, you have to have those the correct tooling to be honest.	
MN	OK.	
DM	And again, that's in the implementation phase, but all of it in the implementation phase and all the way up the right-hand side.	
MN	Umm.	
DM	So, all across the right-hand side.	
MN	OK, alright. So uh, looking at the software unit testing there, would you comment on the practices included there.	
DM	Yeah, I don't see any customer input at that point. The same way all the way up with the exception of the system testing. So, when we are doing system testing, it's actually very good to get some customer feedback at that point.	
MN	Hmm. OK.	
DM	So, we would host things like customer days where people can come in and try our systems and then they try and break it and that's great.	
MN	Hmm.	
DM	There's actually nothing better for my system level to have an untrained user try and break your system to see these, you know, because again, we have an idea of how the flow should work and all that kind of stuff. And people that don't you know, they can sniff out bugs and screens and you know, workflows, incorrect workflows and stuff like that.	
MN	Mm-hmm.	
DM	So, it's very powerful.	
MN	OK. So, you are suggesting that at software unit testing no onsite customer/proxy.	
DM	Yeah, at software unit testing? I don't know customer. I wouldn't see a customer input there.	
MN	Ok.	
DM	Definitely refactoring. Definitely pair programming but no onsite customer there the next the next phase, then the integration testing.	
MN	Mm-hmm. OK. Mm-hmm.	
DM	So, for integration testing, pair programming, yeah, it's quite similar. Yeah, that's OK. I think that's fine the box that you have there, I wouldn't.	
MN	OK. Umm, go ahead.	

DM	Yeah. Again, I wouldn't have the customer here either at the integration testing phase. The customer would be at the next block, the next circle up through the system testing and the acceptance testing.	
MN	OK. What about on unit implementation onsite customer there or proxy?	
DM	No. I don't. I mean there could be very, very early. I mean, it's possible, you know, really, really early when you're just, you know, you have some sort of an early workflow or a POC done that you might want to show a customer just to see what they think. So, it is possible, but it's unlikely.	
MN	Ok.	
DM	I mean, you can leave it there.	
MN	Ok	
DM	You can leave it there just to think, but again, at this point, if you were bringing a customer in during the implementation phase or getting customer input, it would be only about input like do you like the colour of the screens? Do you like the? You know? Doesn't make sense. You know at that point, so you can leave it there maybe.	
MN	Mm-hmm. OK. Alright. And the user acceptance testing. This is at the integration and the system testing.	
DM	Yeah, I mean user acceptance testing is probably at the acceptance testing phase. I mean, it's definitely not at the software integration testing phase and it's further up.	
MN	Hmm.	
DM	So, it's more verification and validation up the top.	
MN	OK. Hmm. So, uh in terms of the structure now, is there anything that you would suggest maybe to change to improve the structure of the framework?	
DM	Umm. No, I mean, I think it makes sense. I think it does make sense.	
MN	OK.	
DM	I see it a bit different, you know. Yeah, I think it makes sense. I think once we update those blue.	
MN	Mm-hmm.	
DM	Uh like if you wanna update them and we can meet again, have a have a chat at some point later on and if that makes sense.	
MN	OK, OK. Yeah. So, looking at the current procedure that you follow in developing the software there and looking at this framework, yeah.	

DM	Uh-huh.	
MN	Do you think that this framework would work within your development?	
DM	I think, yeah, I think it could work. Yeah. I think with some of those tweaks were making.	
MN	Yeah. OK.	
DM	What I would like to see Misheck to be honest with you and this is speaking from someone who has worked in medical devices for a long time.	
MN	Umm. Yeah, yeah.	
DM	I wanna see 62304 and I know, and I know other people because you know you are gonna put this in front of the systems engineers.	
MN	Mm-hmm.	
DM	You are gonna put it in front of people, and they are gonna say, oh, I wanna see 62304 here. So, I would like that, and I mean it's quite similar to 62304 but let me share my screen again.	
MN	Umm. Mm-hmm. OK, alright.	
DM	And my concern is, it's not a concern. Like everything we do, is in the framework. Yeah. Sorry. Can you see my screen?	
MN	Yeah.	
DM	Yeah. So, everything we do is in the framework of the quality management system. It's configuration change management that's across all the project and then risk management and safety processes.	
MN	Mm-hmm. OK.	
DM	They need to, you know, so, just I'd like to see those called out in the framework as well.	
MN	Hmm.	
DM	So, you could just put two boxes either side and two on the bottom as well.	
MN	OK, OK. Yeah.	
DM	That kind of because everything we do is, you know, quality, QMS, the most important people in the in our world are our quality engineers that you know, keep us in check and make sure our projects are you know, are going to get submitted because like we can't go to the FDA with something that hasn't followed this process.	
MN	OK. Umm. Mm-hmm.	

DM	So, it just it won't work with the FDA unless they see that this is, you know, you know this is this has to follow I you know 623 or 4.	
MN	OK, OK. Umm. OK.	
DM	That's the only thing I would like to see that your model to kind of to kind of make sure it's the more successful it'll be is you know it will be as best as how closely it's coupled with 62304.	
MN	Alright. Yeah, yeah. OK. Thanks, D for that.	
DM	No, no problem at all.	
MN	Yeah. And then the other question that I would ask is uh, I know that you are busy and if the possible would it be possible maybe just to have a desk-based implementation of this framework as a means of validation.	
DM	Yeah.	
MN	Once I have done those changes may be we could set a day where you are free and then may I can come and may be just have a desk-based implementation, looking at the past project that you did to say if may be we implemented this would we maybe reduce the risks of defects and the like?	
DM	Yeah. I mean, let me talk to my manager about that Misheck. I know Lee is very, very open to us partnering with external people, external you know research and all that kind of stuff.	
MN	OK, alright. Mm-hmm. Umm.	
DM	We wanna support that as much as we can, to be honest.	
MN	OK.	
DM	And just so I we should be able to do something, it's just I have to check with my manager.	
MN	Yeah, yeah.	
DM	I don't want to tell you, I don't want to say yes, if Lee tells me, it's not possible, does that make sense?	
MN	Yeah, that's good. Yeah, I really appreciate David for your dedication to help in all these processes. So, I would appreciate if you can ask the manager and to see if there is that opportunity that we can have this desk based implementation, yeah.	
DM	Absolutely perfect. OK, Misheck. I will follow up with that and I will come back to you. I am actually on holidays for the rest of the week and next week, so it could be the following week is that makes sense?	
MN	Oh OK. OK. The that would be Ok.	
DM	I am back to week at a 19th.	
MN	19th OK. Yeah.	

DM	Yeah, alright.	
MN	So, uh, I will then send you an email just to get in touch.	
DM	Yeah. Perfect.	
MN	Yeah. Yeah. Alright.	
DM	OK. Thanks, Misheck.	
MN	Thanks so much, D.	
DM	Have a nice day. Mind yourself.	
MN	Yeah. OK. You too, thanks. Yeah, bye.	

E.3 Interview 3

Position: Quality and Regulatory Specialist

Experience in MDS: Over 15 years

Interview date: 30 May 2024 1:30 to 2:38 pm

Speaker	Transcript	Notes
MN	Thank you for accepting to participate in my research. Your insights are very important.	
NS	No problem.	
MN	So, the format I propose is that I'll ask a question and then you provide your views and the experiences.	
NS	That’s ok.	
MN	Yeah. So, do you want me to give you an explanation about the framework or you are happy to proceed to the questions.	
NS	Happy to proceed.	
MN	OK, alright. So, thanks N. So, we will go through the framework. Let me just put it on the screen here. I hope you can see my screen, yeah.	
NS	Yeah.	
MN	OK. Yeah. So, we will go through the framework, looking at each stage and I would like you to discuss the practice in relation to addressing defects. So, as a framework shows, on the left side it's defect prevention and on the right side is defect identification. So, can we start with the requirements gathering stage?	
NS	I'm not sure what would you like me to discuss in relation to how it works currently in the medical devices industry.	

MN	Yeah. So, you will be giving me your views on the practices in terms of whether they will be effective in addressing defects within the stages where they are included and also giving me your experiences of maybe using them, yeah.	
NS	Ah. OK. OK. Thank you. So, yes, I think in the requirements gathering and just like you've depicted there from the user story, from the use case also from both of those, you know when you do your human factors engineering under 62366, the standard as well as following standards. Hopefully companies have standards and have the money to put into buying standards that can be very helpful in identifying risks. Some organizations have a lot of resources and do have a lot of Standards available to them	
MN	Mm-hmm.	
NS	Others are, you know, especially new startups often don't have the resources and are not aware they're not always skilled and competent, so they don't always have all the resources available to them and it's hit and miss.	
MN	Hmm. Umm.	
NS	It is difficult to get software medical device expertise, especially in Ireland. And so, they often don't have all of the full picture in order to address all risks.	
MN	Mm-hmm.	
NS	Now, that said, what's depicted here does provide them the opportunity to do that, it is suggesting that every use case should be considered, and it allows you to prevent defects by appropriately identifying these use cases. So, I think that's very good.	
MN	OK. And in terms of their complementary use?	
NS	When you say the complementary use, what do you mean? Sorry.	
MN	Like using them together, for example using user story and onsite customer in terms of preventing defects and using use case and onsite customer to prevent defects.	
NS	Well, in my experience, there's so much work happening and so many requirements in software development, in medical devices and user story is not a requirement. So, what might happen, and typically happens is that if somebody comes up with a great idea for having user stories or something like that, and if it's not a regulatory requirement, they tend not to do it because they have enough documentation, and most developers are not interested in documentation.	
MN	Hmm. Umm.	

NS	So, to create user stories, that means documentation. It creates a massive headache, so although the agile approach does use user stories, that would not be if I were in work, if I were in a company right now, the usual thing is, do we need to have it? And you would say to me, well, no, because it's not mandated by a regulatory requirement, which is not the case with use case.	
MN	Mm-hmm.	
NS	Use case is mandated and that's why they will focus on those.	
MN	Mm-hmm. OK, so, if I get to you clearly, you are saying that at this stage you would use a use case to prevent defects.	
NS	Yeah, yeah.	
MN	What about onsite customer or proxy?	
NS	It really depends on who is available, and it really depends on the application.	
MN	Umm.	
NS	If you have somebody in the company who has the experience of being a customer for instance, if they were a doctor in previous life or in the current life, they absolutely would provide an input into the use cases.	
MN	Hmm. Umm. OK.	
NS	Umm, yeah, so you know, these are often done as they are not. I was about to say they are done as risk management assessments, but they are typically very poorly done. It's meant to be cross functional and all of that, but usually the doctor is only used for what it is regulated to do, which is to assess the severity of risks of clinical risks.	
MN	Mm-hmm. Hmm.	
NS	So, you'll use whoever you can. In other words, yeah.	
MN	OK, OK. So, if I get you clearly, this means if the appropriate people can be used as customers or proxy you would prevent defects at this stage.	
NS	Unfortunately, it is difficult, as I said, to get resources and often people are overworked if anything, so it can be a challenge for people to reach out.	
MN	Hmm. Mm-hmm. OK. Mm-hmm.	
NS	So, it really depends on the maturity of the organization and on the understanding of the length and breadth they need to go to get the correct information.	
MN	Mm-hmm. OK.	
NS	So, you are really limited by the resources.	

MN	Hmm, but otherwise what would be your views in terms of using these practices there in terms of defects.	
NS	I think it's idealistic more than more than realistic.	
MN	OK. OK.	
NS	And we do need to aim for the ideal because we need to help organizations see that, you know, to apply these approaches we will get better results.	
MN	Hmm.	
NS	Umm, so there is nothing wrong with being idealistic. But you know, the challenge is limit how it's applied.	
MN	Mm-hmm. OK, alright. Alright, so, what improvement should there really be in terms of agile practices at this stage, would you suggest?	
NS	I think I suppose, it would be interesting. I mean, it's really well laid out for a start. It might be nice to show what is a regulatory requirement, in you know, per the regulations, for instance, harmonised standards and things like that.	
MN	Mm-hmm.	
NS	So, it is in the harmonised standard, the usability engineering, a use case is required. It might be nice perhaps to show that in a different colour, but then to make the case to see what the benefit is of doing a user story on top of a use case.	
MN	OK.	
NS	Why do we need both?	
MN	Umm. OK, so, the practices have been included there as choices for the developers. So, either they would choose one like user story or they would use the use case, or they could do a combination. But as you're pointing out, user story and use case may have similar outputs, but of course they could be different in terms of the way they are used and their purpose. So, yeah.	
NS	Would be nice to see which one is a requirement that you have to do, and which one is optional.	
MN	Mm-hmm.	
NS	But I do understand what you're trying to depict there, and it's really nice. But if I am a small organization, I want to know what I must do in order to get through a regulatory submission.	
MN	Umm. OK. So, you would recommend using a use case at this stage.	
NS	Well, it's not that I would recommend it. It's that I am aware that that is a requirement, a regulatory requirement.	
MN	OK. Alright. So, let's move to the next stage, which is the software requirements analysis stage. Yeah. Can you share your	

	thoughts on using model storming, behaviour-driven development, pair programming, TDD and onsite customer/proxy at this stage to prevent defects. So, you would talk on each of them.	
NS	So, on model storming at the top there for software requirements, typically what is required, you see, is some kind of architecture design. The type of modeling you do is dependent on the type of architecture or system that you are developing. So, it could be a UML model, it could be a data flow model. So, it really depends, and it's not clear from that which type you mean.	
MN	Umm.	
NS	So, a lot of this is often driven by the requirements of the FDA, which stipulate that you have to show so you know, you have to present a detailed design and a design architecture with the support of modelling, they don't mandate what type of modeling.	
MN	OK.	
NS	They just say that it has to be appropriate and clear, clearly understood.	
MN	Umm. Mm-hmm.	
NS	So, I think it's well done. I think how accurate or correct it is I don't think that's that difficult to get right because by its nature, once you start drawing something out in an architecture diagram, you know it's easy for the experts who are developing it to see what's correct and what isn't.	
MN	Mm-hmm. Hmm.	
NS	So, I think that's done pretty well.	
MN	Umm.	
NS	I think transferring that into software requirements is a difficulty, and again, typically organizations are looking at trying to pass the regulatory requirements.	
MN	Umm. Hmm.	
NS	So, transferring what the actual architecture into requirements can be challenging and so that it passes the team for review and again, sometimes it can be so challenging depending on the competence that the review is not adequate enough and there it's just accepted.	
MN	Mm-hmm. OK.	
NS	So, some in my experience, documentation can be poorly done.	
MN	Umm.	

NS	It really depends on the culture of the organization and the resources available to them.	
MN	Mm-hmm. OK. But about the behaviour-driven development terms of preventing defects at the software requirements analysis stage.	
NS	I haven't come across it in medical devices.	
MN	OK. But based on your maybe experience, do you think it would be appropriate to use it to address defects at this stage?	
NS	Perhaps I'm not saying is I suppose I'd like to see the evidence and again. If you know it depends, it's not a requirement, so it can just put more pressure on teams.	
MN	Hmm.	
NS	So, it depends on the benefit and I'm not sure what the benefit is over the existing requirements.	
MN	Yes. So, looking at the defects specifically, say, how can we prevent defects as much as possible? So, by using behaviour-driven development at this stage, umm would it be possible to address or prevent defects?	
NS	When you see usability engineering, which is similar to this is what is used right now.	
MN	Mm-hmm.	
NS	But again, it can be poorly done depending on the resources, and often the software developers are not trained in usability engineering.	
MN	Hmm. Mm-hmm.	
NS	So, if you put in another one that sounds yeah, it's a very agile it's an agile practice behaviour-driven development, right?	
MN	Yeah, yeah, yeah.	
NS	But the same activity. So that means that these organizations would have to be trained in agile practices, and sometimes they are but it would be, in my opinion, better for them to understand how they overlap and how they are the same as for the most part the usability engineering.	
MN	OK.	
NS	Because again, it's the usability engineering that's a requirement.	
MN	Umm.	
NS	This behaviour-driven development, although it's the same thing called different name or similar thing called a different name, is not seen as a requirement.	
MN	Mm-hmm. Hmm.	

NS	So, if you were to, for instance, have a mapping to show that in the usability requirement, if you perform behaviour-driven development, that is a substitute or the same or equivalent to something that is a requirement, that would be very powerful.	
MN	Hmm. OK, so you are you are thinking like to say the practice would be like a requirement for it to be done within the development process?	
NS	Correct. Yeah.	
MN	OK. What about using PP at this stage?	
NS	Well, at this stage period.	
MN	Specifically focusing on prevention of defects.	
NS	Pair programming at this stage. Are you suggesting, to me pair programming is when you are actually developing.	
MN	Mm-hmm.	
NS	So, in the requirements analysis maybe it's what we might call a peer review of the requirements.	
MN	Mm-hmm. Yeah.	
NS	OK.	
MN	So, what I mean there to say programmers should be available during this system requirements analysis so that they clearly understand the requirements before they can start developing.	
NS	Yeah. OK.	
MN	So, early understanding the requirement at least.	
NS	OK. Well, usually it is the software developer, the programmers that are doing the requirements as well.	
MN	Umm.	
NS	Often, I know in large organizations you can have your user requirements specification and then maybe in that case they're not in the user requirement spec.	
MN	Umm. Hmm.	
NS	But once you get into the software requirements analysis, that is, you know, done between developers and the users or the people representing users.	
MN	Hmm.	
NS	So, it would be a typical situation.	
MN	Umm.	
NS	And it wouldn't have a pair programmer. You wouldn't have somebody else checking. Typically, what you would have is a	

	quality software engineer, for instance, and whether or not they are fully competent because quality unfortunately gets a bad rap.	
MN	Mm-hmm.	
NS	Developers don't want to be in quality roles, which is one of the reasons, and if you're a good developer, you don't hold the quality role, you hold the core developer role. So, it's often people in the quality role may not have as much knowledge and that is definitely a failing where they're not using a peer competent programmer to review, and then they don't get the competency improvement or value improvement in their requirements?	
MN	Hmm. OK. So, if that would be the case to say there is a competent programmer there to add to the requirements analysis, would it help to prevent defects.	
NS	Yes. The problem is there is a competency requirement. So, no one is going to admit that they are not competent.	
MN	Ok.	
NS	That's the problem.	
MN	So, yes, so, OK, go ahead.	
NS	So, if you had a peer, if you required, but again the regulation does not require this because the regulation and the industry are very well aware that resources are already slim.	
MN	Hmm.	
NS	To get one person as the programmer on a project is wonderful. To request two there would be a lot of kickback from the from industry.	
MN	Hmm. OK.	
NS	So, yes, competency is definitely an issue. And how do you prove it?	
MN	Mm-hmm. OK. What about onsite customer or proxy?	
NS	Again, that's nice to have or a luxury in some cases, so it will be the same all the way through.	
MN	Hmm. OK. Any suggestions for improvements at this stage in terms of agile practices for preventing defects?	
NS	Umm, I don't think so.	
MN	Thanks. OK, then let's move to the software architectural design stage.	
NS	Mm-hmm.	
MN	I would like to hear your views on using model storming, onsite customer, integration testing, and simple design for preventing defects.	

NS	So, I think the architectural design is definitely an area that can yield a lot of value, and the FDA have identified that in their recent guidance. They have shown a few examples of the level of architectural design that they expect to see.	
MN	Mm-hmm. Umm.	
NS	So, that not only is it communicated so whatever they expect is what will be done.	
MN	Mm-hmm.	
NS	For instance, since they are mandating it, that means it has to go through your team review, which means you are more likely to communicate clearly with your team.	
MN	Hmm. Hmm. OK.	
NS	So, the better the architectural designs are, and the more that it's regulated, the better the opportunity for defect prevention is.	
MN	Hmm.	
NS	So, I definitely think that's a strong one in the model. I don't see it being done very much with the customer.	
MN	Mm-hmm. OK.	
NS	And typically, it's your software architect or your software developer.	
MN	Mm-hmm.	
NS	Ideally, you have an architect that's in the best case scenario.	
MN	Hmm.	
NS	Most organizations don't have architects. They have a bag standard. Sorry to call it a bog standard.	
MN	Umm.	
NS	A one size fits all developer who does all of the steps. So, ideally you have an architect and in some of the larger organizations you would have an architect. So, an architect that could clearly depict, clearly transfer the software requirements into the architect design.	
MN	Hmm. OK, OK.	
NS	So, you wouldn't really, the model storming, yes, of course it's taken into account. But more so because it's already depicted in your software requirements analysis. So, the level before is driving your architecture design as opposed to it being a model storming exercise again.	
MN	Umm. Hmm.	

NS	And the integration testing, we haven't talked about development yet, so I am not sure is this the integration at a concept level rather than testing.	
MN	Yeah. So, the idea is that involving integration testers at this phase, they would be able to pinpoint architectural defects.	
NS	Alright. OK. It's just it's not clear that that was meant to be an integration tester. As opposed to integration testing. I thought why is that testing.	
MN	OK. Yeah. So, in terms of agile practices, the term is integration testing, but it's done by integration testers. So, if you involve them there, they would be able to see what the architecture is and then they would be able to write architecture and integration test cases early and by doing that they would be able to point out some of the things that are not fitting together.	
NS	I think it's too late there. I think they need to be at the requirements analysis stage.	
MN	Mm-hmm. OK.	
NS	Because the architecture design reflects the requirements. So, if you don't have integration testing perspectives at the requirements stage, it's for the most part too late.	
MN	Hmm.	
NS	It is very difficult in medical devices to go back. I know we are talking about an agile approach, but it's still culturally extremely difficult to go back on something that was approved because there's often a lot of pressure on getting documentation approved or getting stages completed. And it's usually related to objectives, people's jobs, people's bonuses.	
MN	Hmm. Mm-hmm.	
NS	There are so many cultural reasons why it's difficult to go back.	
MN	Mm-hmm.	
NS	So, I think if anything, yes, integration review I think would be a better word. I do appreciate you have used testing because it's an agile practice and I thought in agile practice it's because you weren't coding early.	
MN	Hmm. Mm-hmm.	
NS	Umm, but I can be wrong.	
MN	Yeah. Yeah. So, in terms of agile, by involving them here, they would, if they had already maybe written some test cases, at the requirements analysis, then they would refine these based on the architecture now. So, that would also help them to point out things that are not fitting together.	

NS	Just where I come from, agile practice was not introduced through a paper based situation.	
MN	Mm-hmm. OK. Hmm.	
NS	It was very much developing live, so the integration testing was a real test where you would test things out, you know, and it was on the fly. It was very non-regulated.	
MN	Umm. OK. Alright. So, have i missed any practice that would be good to have here in terms of preventing defects, at the architectural design level stage.	
NS	No, I don't think so. I think, again, competency in how to represent these architectural designs and it's important that they understand that they have to represent the environment as well as the software system itself	
MN	Mm-hmm. OK. Alright. Let's move to the software detailed design.	
NS	That's an interesting one, because it's going to be removed as a detailed design.	
MN	Ok.	
NS	So, in 62304 the term detailed design will be removed.	
MN	Umm.	
NS	So, it makes me question whether or not that's a whole big circle there. You know the software architectural design will be all one. Which is interesting.	
MN	Hmm, OK. So, just to have like software architectural design and then unit implementation.	
NS	Well, what you could have here is that one the software detailed design could be your architectural design, and the software architectural design could be your system architectural design reflecting the environment, which then makes sense with integration testing because and it also makes sense with your integration testing over here over on the right hand side.	
MN	Umm. Umm. OK.	
NS	So, it's just an opportunity there for future proofing. It's not the current state of 62304, but it will be there.	
MN	Yeah, yeah, yeah. OK. Yeah, but in terms of agile practices included at this stage, at the detailed design stage, in terms of preventing defects, what would you say?	
NS	Yeah. Umm, that is similar to above. Model storming, there won't be any model storming at this stage. However, I am not suggesting that there shouldn't be. If you have a way of incorporating that that makes sense. Great.	
MN	Mm-hmm. Mm-hmm. OK.	

NS	But I think when you are delivering your detailed design or your software design as I am calling it now rather versus a system design, I do think it's helpful to have your customers review of it to know that you are developing the right thing or that you are planning the right thing in a design before you go into implementation.	
MN	Hmm. OK.	
NS	Yeah. And the integration testing piece, I really think that's intended for a development that is live.	
MN	Mm-hmm.	
NS	You know that has started a lot earlier where you are developing as you gather requirements and as you design, you know.	
MN	Mm-hmm. Hmm, OK.	
NS	So, I do question the validity of that.	
MN	Hmm. And so, you would prefer to use this one (integration testing) when the development has started.	
NS	Yeah.	
MN	OK.	
NS	Well, I think an integration review there is no harm in it, especially if it's earlier at the requirements analysis.	
MN	Umm. OK. Any other practice that I may have missed here, that would help to prevent defects at the detailed design stage?	
NS	I think allowing the implementation earlier with the documentation, so allowing the development to occur earlier is key.	
MN	Hmm. Mm-hmm. OK.	
NS	I think that's why agile works and if we suggest that unit implementation starts at the bottom there, we are simply saying that there is no development all the way along here and that's just not the case. I mean people start. So, I think we are lying to ourselves and the world by representing it this way.	
MN	Mm-hmm. OK. So, what do you suggest?	
NS	Personally, I think we should admit that we are developing early iterations at the same time as required, as generating our requirements and that we are allowed to iteratively update our requirements based on that early development.	
MN	Mm-hmm. OK. Hmm.	
NS	But that we haven't gone into a design freeze or anything like that. I think admitting that would be a very, very good opportunity to prevent defects.	
MN	Mm-hmm.	

NS	Because then the software developer doesn't need to hide away in a room developing or the software team doesn't need to make sure the quality are not allowed in the room, which is often the case.	
MN	Hmm. Mm-hmm.	
NS	I have had people come to me and say N, do you realize we have done this already so it's too late now and me as quality would say, well, I didn't realize it, but I can't change the regulation.	
MN	Mm-hmm. Umm. OK	
NS	So, you can see once they have already developed and put all the effort, and you can see how people don't want to go back.	
MN	Umm.	
NS	So, they don't want you in the room and you don't get to review from a safety perspective what they have done.	
MN	Hmm. OK. Alright. OK, so I understand what you are saying that as agile everything is supposed to start together, isn't it?	
NS	Yeah, which would make those things make sense.	
MN	Umm.	
NS	If you have a simple design and an onsite customer or a proxy and an integration testing that would all make sense. If you actually were allowed to show someone that you have developed something already.	
MN	Mm-hmm. OK. I understand. Alright. So, moving to the software unit implementation can I also get your views on the practices included there.	
NS	So, typically what happens is the coder secretly starts developing early and doesn't tell you until they are allowed to do software unit implementation at which case that's usually when they have finished already.	
MN	Mm-hmm. Hmm.	
NS	Umm, so, then it's a matter of the after the fact. They typically start writing test cases. However, now that we are automating testing more and more, the programmer who has done the coding is usually involved in automating the test, which is really super. But again, that's you know, if you use a bell curve some are still back in the paper based system and some are automated and have lots of resources. So, we need to catch everybody. We need to bring the majority of people along this safety defect prevention course.	
MN	Hmm.	
NS	So, writing test scripts can be very onerous. And again, sometimes they say, well, I am not writing test scripts, you	

	know, the software quality engineer can write them more. You know, it can be tough, a tough situation, whereas automating them is the best way to go.	
MN	Hmm. Hmm.	
NS	Number 1 it provides the opportunity for regression testing. Because usually if you do regression testing, that's not automated, you're going to pick the least burdensome approach you are going to pick a few tests, and you are probably not going to test sufficiently which is usually the case, which is why we allow defects out.	
MN	Hmm. Yeah, yeah. Hmm.	
NS	So, automated and implementation and automated unit testing at the same time would be much more beneficial.	
MN	Umm. Mm-hmm. OK. Any other practice that you feel should be included here at the software unit implementation?	
NS	The code review ideally should be automated too by AI or something like that.	
MN	Umm. Mm-hmm.	
NS	So, the huge opportunity there for proper code reviewing and secure code reviews as well. The coding standards, all of the opportunities there, there's a lot of opportunity.	
MN	Hmm, OK. And the in terms of complementary use of these practices, for example, using pair programming, TDD, model storming, and onsite customer coding standards, and code ownership. Complementary.	
NS	From what I have seen, again, resourcing is an issue if you have the luxury of having pair programming, sometimes it can be very difficult. Different ideas, different perspectives and different ways, it can slow down. It's very hard to think when someone's in your ear, you know when somebody is there. It depends on how well you implement it.	
MN	Mm-hmm. But if they were complemented, would it help to prevent defects?	
NS	No, not necessarily. I don't think it would prevent defects necessarily.	
MN	Umm.	
NS	I think it can be more challenging and in practice and it really depends on how well the pair are suited to working together. So, I think to mandate it or suggest it has to be done, I see many benefits, but I also see the drawbacks. So, I prefer to see paired with an AI system that doesn't have a personality. But it depends.	

MN	OK. Hmm. Alright. Any more comments that you would add on this stage? In terms of preventing defects, as well as identifying defects at the software unit implementation.	
NS	No.	
MN	OK. let's move to software unit testing.	
NS	OK.	
MN	Yeah. So, at software unit testing I would want you to also comment on using unit testing, onsite customer or proxy pair programming and refactoring to identify defects, or to address defects at this stage.	
NS	So, in my head I see a big circle. I'm calling these arrows here circles just so you are aware, but I see a big circle under those three. And I think if that was all a circle, that's how you would have a green and an orange in the software unit testing, you could identify as you go. That's the ideal to prevent defects.	
MN	Umm. OK. So, you are talking about software detailed design, software unit testing and software unit implementation.	
NS	Yeah.	
MN	Ok.	
NS	So, if you admit that they are all one by having another circle on top of all of that, because then it makes sense that your integration testing is for within the environment. You are verifying that it works and, but I just see the opportunity for updating your detailed design when you've gone into unit testing. Whereas that's typically a bad thing in medical devices. You know your unit testing. The idea is the unit testing is meant to pass, so they do whatever they can to make it pass. Even if some of them are cutting corners.	
MN	Mm-hmm.	
NS	And often what they have is lists and lists of anomalies that may or may not get reviewed.	
MN	Hmm. Hmm.	
NS	And I have seen organizations that have allowed systems out without quality or rag or anybody outside of the software department knowing about the big list of anomalies. And they are not risk assessed, often for the safety impact.	
MN	Mm-hmm. Hmm.	
NS	So, if you were to admit again or it to be allowed that the implementation can happen with the software unit testing it would be much more valuable because you then admit that you can update your detailed design as you do your testing.	
MN	Hmm. Mm-hmm. OK.	

NS	That is real defect prevention and identification.	
MN	Hmm. OK. And so, in terms of practices included, you have talked about unit testing, but you have not talked about pair programming and refactoring.	
NS	Mm-hmm. Well, I think you know my opinion on pair programming. I am on the fence. Let's say I am on the fence because it can be good, and it can be bad.	
MN	Yeah. OK.	
NS	Refactoring is you know, to me at least, and I mean, I am not the expert, but to me, refactoring is reusing code and segregating and things like that. So, uh, you know, that can be helpful. Is it a defect identification opportunity? Maybe it is. I am not 100% sure there. And onsite customer/proxy, I am not sure that they have any involvement here at all in unit testing. That's typically done behind the scenes by the software team.	
MN	OK. So, what changes would you suggest at this stage in terms of practices included there?	
NS	Well, you see if you made the one big circle those three at the bottom, then that detailed design already has an onsite customer proxy.	
MN	Hmm.	
NS	So, the customer could then see what changes are happening to the detailed design. If you had those three together.	
MN	Mm-hmm.	
NS	So, yes, the testing occurs, whether it's automated or not, but people are not afraid to say this failed because they know it's agile.	
MN	Hmm.	
NS	They know they're allowed to update the detailed design and that way they are not getting the outside perspective to say, well, actually, that could cause this problem or another problem. So, again, you still have prevention and identification at your software unit testing stage.	
MN	Mm-hmm. OK.	
NS	Because if you take it, if you think about it, if you did put a big circle over there, you have greens and oranges.	
MN	Umm. OK. Yeah. So, you are saying that at the software unit testing, I understand what you are proposing here.	
NS	No. Well, I'm saying that my experience is that they are not involved, and I am not sure how, if it was just software unit testing, how does the customer add any value if it's just a testing situation.	

MN	Hmm. OK. Alright, yeah. OK. Thanks for the suggestion at this stage. Yeah, let's move to the software integration and integration testing. Can you also share your views or opinions on the agile practices included at this stage, user acceptance testing, onsite customer or proxy, integration testing, pair programming, refactoring, unit testing, and continuous integration.	
NS	Uh, yeah, I think. I mean, obviously the integration testing is really important, and it depends on the complexity of the system and whether or not you know what we are considering and really without a really good architectural design, it can be difficult to fully capture what needs to be tested in the integration testing.	
MN	Mm-hmm. Hmm.	
NS	So, often what I see is it can be done poorly depending on the architectural design. I mean in terms of planning. It can be, you know, just see what happens, kind of a situation. Umm.	
MN	Mm-hmm.	
NS	So, then again, because it's a regulated environment and not agile, not in the true sense, they still believe they have to do one they have to at least admit to doing one piece after the other.	
MN	Hmm.	
NS	In terms of the documentation, because the documentation is delivered, so even if they're doing it in an agile way, in reality if they found an issue, it can be difficult to go back to the architectural design, unless of course that is represented and accepted that that's what they're doing.	
MN	Mm-hmm. OK.	
NS	Because remember, all those documents need to get signed off by the team again. So, sometimes things get, you know, shoved into nice, neat boxes. And that's not the case at all.	
MN	Hmm, OK.	
NS	So, I think again another opportunity for admitting true agile practices is a better way to go.	
MN	Umm. Mm-hmm.	
NS	So, in relation to all that you asked for refactoring. I am used to seeing refracturing down at the implementation stage. I'm not sure how it's utilised throughout.	
MN	Mm-hmm. OK.	
NS	Well, maybe it's I suppose that goes back to my point. If it's true agile, you are continuing development. You are continuing to change, but in regulation you are not allowed to continue to change.	

MN	Mm-hmm.	
NS	Your implementation is done as you can see down at the bottom there and nobody has said so to suggest we are doing refactoring.	
MN	Hmm. Hmm. Hmm.	
NS	It goes against the V as it's drawn because the V says that you are doing testing here and it's just testing.	
MN	Mm-hmm.	
NS	So, it says acceptance testing, system integration testing and unit testing. It doesn't say continue development; you know. So, I suppose I find it a bit strange.	
MN	Mm-hmm.	
NS	And regression testing is a real thing, which is what I suppose I would expect to see. And again, it's because unit testing is not admitted to being done but it is in fact done as a form of regression testing to repeat when repeat is different but to identify what test needs to be redone.	
MN	Hmm.	
NS	So, if that said regression testing, it would be and not that it's more understandable, just more reflective of the terms that are used in medical.	
MN	Mm-hmm. OK.	
NS	Yeah.	
MN	Right, then what about user acceptance testing and onsite customer.	
NS	And again, if we have them available to us, I think it can be helpful. But at this point I'm not sure how helpful. Actually, and at this point we're just trying to get the architecture to work together, you know, to make sure that we are able to move into system testing.	
MN	Umm.	
NS	So, really at the integration testing stage, you are expecting the performance from the system and if everything all works together, you know. And it's really integration of the software units themselves that they all talk to each other correctly or is it integration into the environment and that's why I think that software architectural design, if it represented the system architecture would be an interesting shift. I think for some organizations to know whether or not it's integration of just the units or is it integration into for instance, a lot of the issues and now that we have cloud and we have the applications and we have Google, whether or not it's an app that's made available for	

	download. Often, they don't have the ability to test that until very late, so that is causing a big problem actually.	
MN	Hmm. Hmm.	
NS	And I don't know if you are aware of the issue that occurred with Abbott. Abbott had you know that patch for the diabetes prick test to replace pricking the finger to get your blood. They had a patch and last year they were down for four days because they made a change.	
MN	Umm.	
NS	They uploaded it to Apple, and it didn't work. And then Apple were not able to roll back because they have a policy of not rolling back.	
MN	Hmm.	
NS	So, they couldn't just reverse back to the other software system. It meant for four days there were people without their application and there was a lot of it cost millions.	
MN	Hmm.	
NS	So, you can see how Integration testing, again, you're right, there is a huge opportunity to prevent or identify defects.	
MN	Mm-hmm.	
NS	And I do wonder if there's an opportunity there for prevention as well as identification, because it doesn't always have to fail for you to realize there's going to be a problem.	
MN	Umm. Yeah. So, we are saying it's identification because here the units have been developed and we are integrating them. So, when you are integrating, then you identify what is working or what is not working. If they are not working together, then it means you have identified the program which needs to be fixed or to be corrected, yeah.	
NS	Yeah. Yeah, but there is, I would hope, still an opportunity even, I suppose if it's really agile, you are still preventing.	
MN	Umm. Umm.	
NS	So, I could have realized. OK, I am now going to use Apple. I know that they can't roll back, so in order to prevent that I need to do something, you know. So, I am just questioning the opportunity to prevent as well as identify.	
MN	OK. Hmm. OK. Yeah, I think that's a good point. It would really, I think help to reduce the risk of defects.	
NS	Yeah, I think so too.	
MN	Hmm. Hmm. OK. So, we move to the software system testing stage. Here, I would like to hear your thoughts about using pair programming, unit testing, refactoring, and also user acceptance	

	testing and onsite customer/proxy in terms of addressing defects at this stage.	
NS	I think, well, it doesn't entirely make sense to me why we have user acceptance test at both the acceptance test and the system test. Having the customer included throughout the process is obviously ideal, but in medical devices, and I do appreciate you have said proxy so it's someone standing for them which is good.	
MN	Hmm. Umm.	
NS	The system test, so again I think it comes back to the requirements and knowing that you can update the requirements at this stage and if you do, what's the impact to the architecture design and the detailed design, the unit implementation. So, yeah, I suppose, I mean, I think you have reflected it well visavis.	
MN	Umm. Hmm.	
NS	And the pair programming, I'm not sure. Well, I suppose that I'm not sure how that benefits and like I'm on the fence with the PP, so we'll leave it there. The refactoring does make sense as in terms of segregation of the system and but is it suggesting that we're still changing code.	
MN	Yeah. So, at this stage, yes, we are changing some of the things that would crop up here.	
NS	Hmm, so it seems not aligned to medical device system engineering or design control. Maybe the answer is that, well, I don't have this answer, but I think the only way this can work is if we get rid of design control and that's not gone. That's not to say that you wouldn't be controlled, but it's just if you start changing unit testing here and refactoring code at the system level, you really do need to. It's a huge design change according to design control, so I'm not sure it would be accepted.	
MN	Hmm. Hmm.	
NS	It's a design change. You have to reassess everything.	
MN	Yeah. So, the proposal is that the if the defect identified here does not require you to make a lot of changes on the other the stages, then it can be corrected there and that's it. But if it is a defect that requires to go back, then you need to go Indeed, they're changing the design.	
NS	But any change is going to change the version, right?	
MN	Yeah. So, sometimes you might for example here the system level, I give an example. Maybe the button is supposed to be showing maybe green, but it doesn't so that would not require you to go back and change everything down there.	
NS	Yeah.	

MN	You just make that small change there and then things would work.	
NS	Yeah, but you still have to demonstrate that that is a controlled change that everyone in the team agrees to that change, and everyone agrees the impact that it could have because it might be that marketing wanted that green and it particular type of green.	
MN	Umm. Umm. Hmm.	
NS	So, that needs to go back through an assessment. So, yes, I know you can make it, but it still needs to go through a design change assessment because you are changing something.	
MN	Mm-hmm.	
NS	Plus, it will have a change of configuration. The documentation needs to change.	
MN	Umm.	
NS	Maybe the I would imagine the documentation and the software version needs to change. So, I don't know. It's difficult to see how it can work through agile can work in design control.	
MN	Ok. Ok	
NS	Opportunity for a paper, I think. Definitely because developers are saying this similar thing to me.	
MN	Umm. Umm. OK. Anything more that you would like to add in terms of defects at this stage and add agile practices?	
NS	No.	
MN	OK, then the acceptance testing stage. So, I would also like you to hear your thoughts here in terms of agile practices included, user acceptance testing, pair programming onsite customer. What are your views in terms of identifying defects or addressing defects at this stage?	
NS	Just if I may, I was cognizant of the time and I was trying not to say too much, but because I say a lot and I one thing I think is important because it's allowed in the Reg, the software system testing, and the integration testing is often done together. Just so you are aware.	
MN	Hmm.	
NS	And so, they combine those two things together. So, what gets tested gets diluted and that can yield its own opportunity for defects that are not seen, and so then acceptance testing.	
MN	Hmm. Hmm. Hmm.	
NS	Typically, the software system is handed over and they don't worry about it too much anymore.	

MN	Mm-hmm.	
NS	Because it has to go through the regulatory. So, those three are I suppose all one. They don't typically talk about acceptance testing. The top level is system testing and integration is combined.	
MN	Mm-hmm. Mm-hmm.	
NS	So, in my world, we combine all of those together and they are verified and or validated. Doing the whole thing together means that it's valid, validated system and if whether or not you have a clinical investigation or clinical study that can be used as your validation. Often, they do a usability study which they call the validation, so they give it to some users as part of the usability and they then call that the software validation.	
MN	OK. Hmm. Umm.	
NS	But the opportunities in that software validation. So, any usability study for validation study only needs to identify critical failures.	
MN	Umm. Hmm.	
NS	So, again, there's another opportunity to do a proper usability software usability study rather than the one governed by IEC 62366. I have just forgotten the number for somewhere reason, never forgot that before, but anyway and still I don't know if I see the currently.	
MN	OK.	
NS	If we did all of it, I would be wondering how it differs really, and what actually we are doing.	
MN	Mm-hmm. Hmm. OK. So yeah, uh, talking about the three practices included there.	
NS	Yep, yeah.	
MN	OK. So, your commendation would be we use one or what would be your suggestion there?	
NS	My suggestion is not to make it one, I think. I think the integration testing is really, really important to be separate and as it's shown, but it it's really, I suppose depends on the complexity of the system.	
MN	Hmm.	
NS	Sometimes it does make sense to make them one, which is why the Reg allows it, but because they are allowed to be one, you can imagine it's human nature.	
MN	OK. Mm-hmm.	

NS	They say, oh my goodness, great. We only need one document because it's not about the testing and it's, you know, it's not about the development and it's about getting over the paperwork.	
MN	Hmm. Umm. OK. Yeah, yeah, I understand.	
NS	Yeah.	
MN	OK. So yeah, having looked at the framework, yeah. Is there anything that you would suggest in terms of improving it? I know we have had the discussions before, but I still need your ideas.	
NS	Umm, I think the requirements gathering to suggest the acceptance testing.	
MN	Umm.	
NS	Maps to the requirements gathering. I just have that. It's an interesting way to look at. I suppose when I look at it here today, I see that the validation is done to the medical device. If I am thinking about design control and it's difficult for me not to think about design control.	
MN	OK.	
NS	So, I suppose the one thing I would think would be amazing would be to sit down with the regulators like the FDA and to challenge them about design, control and agile and see what they think, you know, or the people who wrote the TIR 45. To truly understand where they're coming from.	
MN	Umm. Hmm.	
NS	Umm, I think that would be highly beneficial to get their input on it.	
MN	Hmm. OK.	
NS	So, I don't really have any other comments, though I don't know if that's enough.	
MN	OK, so uh, looking at the framework and the IEC62304 standard, if someone was to adopt this framework, would they be conforming to that standard?	
NS	So, they would, because that standard suggests a V-Model that starts with the requirements. It goes into the design, goes into implementation, then testing.	
MN	Umm. Umm.	
NS	So yes, absolutely they would.	
MN	Hmm. Hmm.	
NS	Plus, it's also a process standard that shows best practices but doesn't tell you what model to use and doesn't tell you what development approach or lifecycle methodology to use.	

MN	Hmm. Hmm.	
NS	So yes, in short, yes.	
MN	Umm. OK.	
NS	It has extra stuff.	
MN	Ok. So, do you have any other comments that you would want to make?	
NS	No.	
MN	OK. All right. I think we have had our discussion, and I think the questions that I was supposed to ask you, we have exhausted them. And I just want to thank you for sparing time to have this discussion and I look forward to having more discussions with you. I would love if you I would like to be contacted again.	
NS	Absolutely. No problem Misheck.	
MN	OK. Yeah. So, thanks so much for the time and the insights that you have shared.	
NS	My pleasure.	
MN	Yeah.	
NS	Good luck with it.	
MN	OK. Thanks so much.	
NS	Take care. Bye-bye.	
MN	Yeah. OK. Bye-bye.	

E.4 Interview 4

Position: Software Engineer, Tester & Project Manager

Experience in MDS: Over 10 years

Interview date: 6 June 2024 2:00 to 3:08 pm

Speaker	**Transcript**	**Notes**

MN	Hi AC. First, I would like to thank you for accepting to participate in this research. You contributions to this research are valuable.	
AC	No problem.	
MN	Before we proceed with our discussion may request your permission to record our conversation.	
AC	Yes, you can record.	
MN	Thank you for allowing me to record.	
AC	That's fine.	
MN	Yeah. So, I will share my screen. I hope you are able to see my screen.	
AC	Yeah.	
MN	So, what you see there is the Agile-based Defect Addressing Framework which has been developed based on the information gathered through the literature review and the survey of medical device software developers.	
AC	Yeah.	
MN	So, the main aim of this framework is to help medical advice software developers to address defects by first preventing them in the early stages. So, prevention happens on the left side of the framework there. And then also to identify defects that may slip into the later stages. So, identification happens on the right side of the framework there. So, the idea is that at the stages on the right, uh developers should use the proposed agile practices to prevent defects. So, for example, at the requirements gathering stage and uh, they will have user story, onsite customer, and the use case. So, the framework provides the flexibility for developers to select either one practice to use at that particular stage or multiple practices. In the case where all the practices are selected, for example at the software requirement requirements analysis stage, we have provided the best practice there to say OK, you should start by implementing model storming and then behaviour driven development and then this should be followed by pair programming, TDD and onsite customer so that you enhance the defect prevention at that stage. So, similarly on the identification part, uh, we have also provided the best practice in using the agile practices at certain stages. So uh, the reason for contacting you is to get your views on the framework in general in terms of addressing defects. So, prevention and identification, and also suggestions on how this framework can be improved. Yeah.	
AC	Yeah. Um.	
MN	So, the proposal on the way I want us to progress is that I will ask you a question and then you provide your views and also experiences where applicable.	
AC	OK, that's fine.	
MN	Yeah. So, we proceed like that.	
AC	Ok.	

MN	Yeah. So, we will start with the requirements gathering. So, at this stage we have user story. What would be your views in terms of using user story at this stage to prevent defects?	
AC	Yeah, I mean this is a fine technique, I guess. Umm. There is no problem I see with this, you know. These are all fine. You know, essentially fine techniques to do what you're trying to do, which is gather requirements. The main thing here um is the steps that are missing in this piece from the real world's kind of medical device software development is how it fits into. It's not that, uh, this wouldn't happen. It's that there will be some additional steps.	
MN	Hmm, OK.	
AC	Namely, it's how it fits within a larger sort of quality management system that is also addressing risk management within this development.	
MN	Mm-hmm. OK.	
AC	So that's uh, I mean, that's fine.	
MN	Hmm.	
AC	It's fine for me to say that it doesn't invalidate this structure. And you were maybe surprised to find that probably most respondents used some sort of V-Model in development.	
MN	OK. Umm. Mm-hmm.	
AC	I don't know if that surprised you or you thought, OK, that it was, was that expected or not expected?	
MN	Yes. So, from the literature review, uh, I think it shows that most Yeah, organizations use the V-Model.	
AC	Yeah, but do you understand why? And from medical software?	
MN	And I think it's because of the requirements for the regulators, or?	
AC	That's essentially it. I mean the main thing that differentiates regular software development from medical device software is just regulation and certification.	
MN	Hmm. Mm-hmm.	
AC	So, when you are trying to meet the requirements of the different standards that you need to meet. So, for example 62304, you are basically which is, you know, basically how you develop and document the process for making software in a medical device.	
MN	Hmm. Hmm. Umm.	
AC	They are very kind of rigid in looking for Inputs to outputs and the V-Model maps almost directly to that.	
MN	Mm-hmm. Mm-hmm.	
AC	There's absolutely um, latitude to apply agile methods within each of these phases as you've done. I think that's absolutely possible. Uh, so I think you're on the right track here with having agile methods within this V-Model shape. That's that would, but that is the first thing that the first verification I would have, or I would suggest for yourself is to see is to double check that this.	
	Ok.	

AC	I can point you to a number of resources to help you do that if you'd like.	
MN	Mm-hmm. Umm.	
AC	I don't know how familiar you are with these things.	
MN	No, you can provide the resources, yeah, so.	
AC	Yeah. But in any case, you have identified a number of tools for doing requirements gathering.	
MN	Hmm.	
AC	All of these things are good and there are additional things that would play into.	
MN	Mm-hmm. OK.	
AC	Medical device software, just by a standard, for example, it should be consistent with the state of the art.	
MN	Mm-hmm.	
AC	In terms of, not just in terms of technology, but in terms of the medical device, uh, creating care itself.	
MN	Hmm.	
AC	So, for example, you wouldn't use out of date techniques for creating the medical device.	
MN	Ok.	
AC	Umm by that I mean uh, let's say you're doing some ECG (electrocardiogram) analysis. Your ECG analysis wouldn't be looking for patterns that are no longer scientifically relevant.	
MN	OK, by ECG, you mean?	
AC	That just as an example, you are doing some signal analysis on the electrocardiogram of a patient. You know, as an example of medical software.	
MN	Umm. Hmm.	
AC	But the standards will give you an idea of the sort of those baseline things that would also feed into uh meeting a business need or a customer need, you know.	
MN	Hmm.	
AC	So, those are the two other things. So, all you know, obviously user stories here are important to help capture the customers or the stakeholders requirements.	
MN	OK. Hmm. Mm-hmm. OK. what about the use case at this stage?	
AC	Yeah, this is all good. Yeah. So, these are all appropriate methods for these gathering requirements.	
MN	OK, so specifically looking at the defects.	
AC	Yeah.	
MN	Would they be appropriate to help to prevent defects at this stage?	
AC	So, I guess, uh, yeah, I mean, yes, anything that provides extra detail at this point would be appropriate. Like so I don't know what and what are you considering as a defect.	
MN	Yeah. So, like for example at this stage would be maybe ambiguously requirements or you fail to capture what the customer really requires because maybe you did not include the customer. So, if you include the customer at this point, do you	

	think that they would help to at least the uh provide clear requirements and that in essence prevent some defects that would be incurred later.	
AC	Yep, yeah, for sure. So, say take our product, we didn't have it wasn't commissioned by a specific customer.	
MN	Mm-hmm.	
AC	We're creating a product and we're hoping that it will be used by kind of the intended audience.	
MN	Umm.	
AC	So, we take a sample from the from those intended audience and we describe or, you know, we describe our ideas for the product. They come, they give us some feedback and that's an iterative process and like that you can do.	
MN	Umm. Mm-hmm.	
AC	So, you are capturing. You are using user stories here or and then you are making you are creating use cases to verify, and that's kind of an iterative process as customers that absolutely helps prevent defects.	
MN	Umm. Hmm, OK.	
MN	OK, now moving to some maybe improvement that you would suggest here. To say, maybe you missed something in terms of other practices at this stage.	
AC	Yeah. So, we don't explicitly use the agile practices, so I can't really tell you if you're missing some agile practices.	
MN	OK. Ah, OK, OK.	
AC	I can really just give you some feedback on the dev side of things and let you know sort of the additional pieces that would go into this.	
MN	OK. All right. So, let's come back to that, to the V-Model bit later.	
AC	Yeah	
MN	But for now, I would like if you have any views on the other practices included on the stages and then later on you can give me the overall use in terms of the V-Model.	
AC	Umm.	
MN	So, I would like us to get to the second stage, which is the software requirements analysis. What would be your opinion in terms of using model storming or uh, be uh behaviour driven development, PP, TDD, and onsite customer at this stage where you are doing a requirements analysis in terms of defect prevention.	
AC	Yeah, again, uh, they are all fine.	
MN	Umm.	
AC	I don't see any problems with that. At this stage, like at this stage they, you know, they're all achieving the aim of basically telling you, giving you some more detail in your requirements, right.	
MN	Mm-hmm. Yeah. OK. So, on software architectural design. The practices included at that this stage. What would be your opinion in terms of preventing defects using these practices? Model	

	storming, onsite customer, Integration testing, and simple design.	
AC	Yeah. I mean there. Uh, depending on the medical software, for example, the onsite customer may not be possible.	
MN	Umm. Mm-hmm. Umm.	
AC	It may be that, so this is just a kind of an insight, it's just that it may have not been possible until the uh, full development cycle has been complete. Or until it is, uh, passed, like software system testing.	
MN	Mm-hmm.	
AC	Umm. And that's simply just there may not be any resources at the customer to provide that, but also, they are kind of managing an amount of risk in terms of how they might use	
MN	Umm. OK. OK.	
AC	But like certainly customer feedback at any time is great	
MN	Hmm. So, you are saying that the constraint there would be, if I got you correctly, it would be in terms of resources, in including the customer.	
AC	It essentially, it is yeah. Like uh, depending on who the customer is for the medical device, if it's used directly with a patient, maybe you can get a patient.	
MN	OK. Umm. Mm-hmm.	
AC	Like it will be difficult to use it directly or get direct patient feedback at such an early stage.	
MN	OK.	
AC	Uh if it's used, say in a clinical setting, you will be able to get some feedback. But possibly, uh, at this stage the product will not be well, enough understood. Or in a form that can be done on in an onsite. That's all I am just cautioning against.	
MN	Umm, OK alright.	
AC	So uh, so I will say again, just, uhm, sort of the one of the main things with the medical piece of the software is this idea of risk management, so there should be a piece coming out of each say out of requirements gathering out of analysis out of architecture that essentially is trying to do identify risk at each stage. And so, for example, if you create an architecture for your software system, no matter what that architecture is, there will be something like communication and opportunities for communication failures.	
MN	Mm-hmm.	
AC	Like even if you have a perfect design system, it's very difficult to prevent communication failures and so these sort of so risk analysis at this stage kind of will then feedback into requirements analysis and architectural design.	
MN	Umm.	
AC	And it's important to know that those are already requirements when doing software medical device software.	
MN	OK. So, you are suggesting that at each stage there should be an element of risk management.	

AC	At some stage it would be appropriate to do hazard analysis and at other stages it will be appropriate to do, uh, failure modes analysis.	
MN	OK. Hmm.	
AC	And uh, at all times, at every stage, risk management should be considered.	
MN	OK	
AC	That's the piece that would make it truly medical software. That's what would differentiate it from being just normal software development with less defects.	
MN	Hmm.	
AC	A company is under an obligation to reduce risk as far as possible. And that's in its own practices you know for managing the company but also specifically through implementing the software development procedures that comply with the various standards.	
MN	Hmm.	
AC	You also have to comply with 14971, which is Uh, yeah, just risk management at every level.	
MN	Yeah.	
AC	But yeah, this still would hold like this General shape. These General techniques will all hold within this environment.	
MN	Hmm. OK. All right. Yeah, I think this is a really very important point you have just raised there. Umm so, uh we have looked at the software architectural design and what would you say about agile practices included at the detailed design stage. There is model storming, onsite customer, integration testing and simple design.	
AC	Yep, absolutely. You know all of those things are great. Just any anything that gets you closer to fully understanding the implications of your design and your requirements is great, yeah.	
MN	OK. And moving to the software unit implementation. What would be your views in terms of the complementary use of agile practices at this stage?	
AC	Yeah, I think they are all good. The main thing essentially as part of managing the lifecycle. There tends to be an overarching plan in place for these things, but I don't know if that is actually required in this case. I am just trying to think if it's mandatory or not, but in any case, these techniques are great. Coding standards all help towards defects minimization. And there are specific. I don't know if you have looked for any specific medical software coding standards.	
MN	No, but well, would you shed some light on that?	
AC	Yeah. So, one of the things is to look at big companies. What we have done, because we are a small company, we have looked at big companies that do have published coding standards and we have kind of used them as an industry point touchstone.	
MN	Mm-hmm.	

AC	So, for example, Phillips has a number of coding standards for different languages that are pretty good practice.	
MN	Umm, the name again you said.	
AC	Phillips	
MN	Oh Phillips.	
AC	Umm yeah, so I don't know if I can I share links here? I probably can put them in chat.	
MN	Yeah, yeah.	
AC	Well. There is Phillips and then there are things like. I will have to look at possibly some other ones, but like you could look at some new ones which are like, OK, so if you're doing uh, cloud for healthcare, Microsoft have a bunch of design principles.	
MN	Hmm.	
AC	So, at this unit level what you would be doing is, you know, trying to apply these as best you can and then either as your unit test level or like so you would decide how you were going to verify whether or not you did follow those things.	
MN	OK. Mm-hmm.	
AC	So, you know you have code reviews to do that, you know to make sure you are complying with your coding standards you have. And you know your best practices and design. Of course, all of these other things help, unit testing, pair programming and continuous integration. All helps.	
MN	Ok.	
AC	The thing that kind of hurts Agile in the medical space is that it tends to, uhm, you know, you are talking about a small release happening at unit implementation. But that probably doesn't get out to a customer at this level.	
MN	OK. Hmm.	
AC	Depending on the type of change, you know, our findings so far is that basically like certainly for the first version of a product, you follow all of these steps, and you know some of the steps, some of the circles you are going around multiple times that's true, but every change almost requires like every single step. Again, we don't get to skip steps in the future and that's why we don't really rely on like any pure agile processes. If you know what I mean.	
MN	Hmm. Hmm. Hmm.	
AC	So, what ends up happening then is the thing that is released at the end needs to be fully traceable. And then you have to have all of your verification done.	
MN	Umm.	
AC	So, you identify the types of verifications that are needed for that change, and you've already assessed the change for the risk management of the change.	
MN	Hmm. OK.	
AC	But what it amounts to is essentially doing a full V-Model.	
MN	Hmm. OK.	

AC	So, you yeah, it's just hard to think that you could skip any of these steps, if you know what I mean. For any numbered, any specified version of the software.	
MN	OK. So, I think you have talked about unit testing there in detail and what about the other practices there, in terms of the addressing defects. I think you have talked about code review and unit testing and small release.	
AC	Yeah, all of these are very important as well, like refactoring along with continuous integration, so that you are ensuring that anything that you do change doesn't have any unintended knock on consequences.	
MN	Hmm. Yeah, yeah.	
AC	And so, you know, there's always tests for regression beyond the scope of any changes that you are making?	
MN	And now let's move to the SW unit testing stage. There we have unit testing, onsite customer, pair programming, and refactoring to address defects at this stage. What would be your views on using these practices?	
AC	Umm. Yeah, it's all good. Essentially, at this stage, you know your software unit is complete, you run your unit testing. That's either a pass or fail type scenario. If it's failed, you are going to do some change, some pair programming, or some other type of programming to fix. If the tests are sound and the unit is like fundamentals are fine, you are going to do some changes in running unit tests again. I think this is all fine.	
MN	Umm. OK.	
AC	Yeah, I mean essentially you have captured at every stage here in the process the things that can be done and should be done, you know in this order as well. The ordering is important.	
MN	Hmm, OK.	
AC	Like I your integration testing is next. Obviously, that's grand. That's fine. I don't have any problems with this. And uh, you know, it's the same process. You know, you find the problems you address the problems if they can be done within that cycle. Otherwise, you know, you have a fundamental problem. Uh, and again at the system level.	
MN	Hmm.	
AC	The only thing that I would say at acceptance level is there is one additional standard that might be considered.	
MN	OK. At the acceptance level?	
AC	At the acceptance level.	
MN	Ok.	
AC	So, you already have user acceptance tests. But there is a standard called the 62366, the IEC 62366 and this is usability.	
MN	OK. Yeah, usability.	
AC	Yeah. So, what they would talk about there is essentially for any product you would define a usability evaluation plan. Then you would develop the usability evaluation protocol.	
MN	Hmm.	

AC	Then you create your scenarios. But it's really the usability of the system that's what they are focused on rather than any other aspect of.	
MN	Ok.	
AC	They are not concerned with anything else that the device is doing, is just can the users use it.	
MN	Hmm. OK.	
AC	So, that's it's just another thing to consider. It's like another input.	
MN	OK. Hmm.	
AC	And especially the first iteration is very important and then you would consider whether or not you have changed the usability at any other like I under change management thing.	
MN	Hmm. OK. So, coming back to the general feedback, you said that I think you mostly use the V-Model.	
AC	Yeah.	
MN	So, looking at your current methodology and then using maybe together with this one, what would be your opinion in addressing defects?	
AC	So, one of the reasons we don't use kind of these, some of these techniques is because we are a very small company. We find that some of the agile methods add an overhead that is not worth it at our size.	
MN	OK, OK. Umm.	
AC	So, I can't speak to the effectiveness of each of those techniques, but I think you have captured essentially all of the steps that we would be doing within each of our processes here. All of the things that we would consider.	
MN	OK. OK. So just to just to ask again, and would you be able to adopt this framework in your organization?	
AC	Yes. Yeah, we would.	
MN	OK.	
AC	Yeah.	
MN	Right. Mm-hmm.	
AC	So that yeah, with the caveat that we would have to take this framework and as it's currently written and build in the risk management pieces that are required.	
MN	Umm.	
AC	But essentially, all of these uh pieces would be completed, yeah.	
MN	OK.	
AC	The other thing that is important to note is um, essentially at every stage for the medical device, there should be an output, some defined outputs. That, you know, could be tracked or they're identifiable at some later stage.	
MN	OK.	
AC	So, part of that also helps identify if a defect is discovered later, you want to know what part of the process it was introduced, but also you want to know what pieces of software it affects.	
MN	Hmm.	

AC	So, it needs to be traceable from that point of view.	
MN	Ok.	
AC	So, the things that you do for containment, let's say of a defect once it's discovered, it's important to have outputs of for each of these things.	
MN	Umm hmm. OK.	
AC	So, for example, you know, yeah, your software architectural design step will have an architecture document.	
MN	Umm yeah. Hmm.	
AC	You know, it's not very complicated, it's just there is some sort of versioned document that is available to us, like the unit testing will have unit test runs with test reports and what unit tests were run.	
MN	Yeah. OK. Umm. Umm.	
AC	And the same with all of the testing. We will have test outputs. They will have labelled software that went into the testing. They will have who did the testing, when it was done, and all of that like some of those things we kind of they are taken for granted.	
MN	Hmm.	
AC	But at some stage an auditor comes in and says show me the process. Show me the evidence that all of these steps were done and that you followed your process.	
MN	Hmm.	
AC	So, it's important to have exhibits for those type of things. And then in a worst case scenario, an incident occurs.	
MN	OK. Hmm.	
AC	So, imagine there's a defect in the fields where there were some patient harms because you have, you know, this is medical software. An investigation will occur where you need to be able to you know, say what was what? What software is out? How did it get out there? And you know what containment is required.	
MN	Umm OK.	
AC	So that's, I mean some of that happens after design and development, but some of like the things that you do within design and development also feed into this.	
MN	Umm. Hmm. OK.	
AC	Yeah, yeah.	
MN	Alright. Yeah, that's really very important. Umm.	
AC	Yeah.	
MN	Hmm. OK. So, any more information that you would want to provide in relation to the framework?	
AC	What I would say is, if you have a look at the open regulatory templates, I don't know if you can open that link at the moment and share your screen. I will just explain a little bit about why that's important for me.	
MN	OK. Yeah. OK, alright. Yeah, yeah. You can share.	
AC	Yeah. OK, one second. OK. You can see that.	

MN	Yeah.	
AC	OK. So, Open Regulatory just briefly about them, is they create a QMS software. So, quality management system software that we use and but hopefully you can see a bunch of stuff from that software for free and you can actually get in and look at it.	
MN	Umm. Hmm.	
AC	Quality management system is just a way to design your company so that it makes good quality products.	
MN	Yeah, OK.	
AC	So, as part of this, there's an overarching thing which is the MDR, medical device regulation, is the latest European standard that your medical software is going to comply with.	
MN	Hmm.	
AC	So, right now we comply with the old MDD, Medical Device Directive, and we are transitioning to MDR.	
MN	Ok.	
AC	So, in Europe, that's happening at the moment, everywhere.	
MN	Mm-hmm. OK.	
AC	So, as part of that you must be creating your software within one of these environments.	
MN	Umm.	
AC	Before, it used to just be ISO 13485 compliant environment which just says you need to have a bunch of procedures for a bunch of different things within your company.	
MN	Umm.	
AC	SOP being standard operating procedure, so you if you looked at all the templates, you would get an idea for all of the different things that you have to have in place.	
MN	Yeah. OK.	
AC	So, clinical evaluation, corrective and preventive action, all of these things, right. So, for example, you take feedback management and customer complaints, what you would see is that's just how to uh, it's just telling you how you should get or capture complaints and feedback, right. It's how you do it within your company, but it also might tell you what happens after you have captured those things.	
MN	Hmm. Ok.	
AC	So, that would then feed into for example, in your diagram like the requirements gathering.	
MN	Hmm.	
AC	OK. This is all fine, but it's just like OK, you have a bunch of things to do here.	
MN	Ok.	
AC	And that's before you even get to specific software development.	
MN	Hmm. Ok.	
AC	OK, so software development then fits in a particular place within the overall quality management system.	
MN	Hmm. OK.	

AC	OK, so, the main thing here is this SOP Integrated software development, right.	
MN	Hmm.	
AC	So, what this will display to you. You can see this PDF, right?	
MN	Yeah, yeah, yeah.	
AC	So, at the start of the PDF, what it's saying is, which sections of the standard is covered in this document, and you can see it's covering a bunch of different standards. So, that's just a little, that's just an index you can looking it up, but here what you're gonna see described is basically the process steps for a generic software medical software company.	
MN	Hmm.	
AC	So, you have your design inputs where they come from.	
MN	Hmm.	
AC	So, in your case you would take this example template and you would add in your agile methods here.	
MN	Umm. Hmm, OK.	
AC	And so, the piece that you're missing at the start at least is the risk management planning.	
MN	Hmm. OK, OK.	
AC	So, you can see that it does take up a bunch of time here, and there's a number of outputs.	
MN	Yeah. Yeah.	
AC	So, this will define a risk procedure that you will refer to, but that risk procedure will basically talk to you, talk about how you create a risk management plan for this product.	
MN	Umm. Mm-hmm.	
AC	How you do risk assessments and usability evaluations. So, the usability evaluation, that's the other standards that I mentioned before 62366.	
MN	Umm. Yeah, yeah, yeah.	
AC	And once all of those things are done right, now you are into software planning.	
MN	Hmm.	
AC	And so here, you know, you start to fill in, you just outline. You know, OK, so it talks about getting requirements and feeding in the risk thing. So, you have your inputs here and output.	
MN	Umm.	
AC	So, this maps to your framework.	
MN	Mm-hmm. OK.	
AC	It's just that I would refer something like this or like the standard to make sure that your framework has the extra little pieces that make that differentiate it to being medical software as opposed to just software.	
MN	Hmm.	
AC	Yeah. Unlike, they are just defining a kind of a generic way that medical software could be made, what you would take is you would say, OK, well in this step you we do agile processes rather than these other processes.	

MN	Hmm.	
AC	You know, they talk a little bit about that here and you can see it's a little bit light on detail in terms of like describing what a company might do.	
MN	Umm. Yeah, yeah.	
AC	But it does give you some ideas, and it also highlights some maybe specifics that you were not aware of like SOUP, software of an unknown Providence.	
MN	Umm. Hmm.	
AC	What that means is basically any software that you are not making yourself, but you are integrating within your product.	
MN	OK, OK.	
AC	So that could include any frameworks or any, you know, any libraries or anything like that you have to do some specific things with those pieces in order to meet the requirements of the standard.	
MN	Umm. Umm. OK.	
AC	You have to establish if there are any known issues with them and any implications, though those known issues would have for your software, things like that.	
MN	OK. Umm. OK.	
AC	And sometimes, like it's very common in modern software development to just use a nugget package and then pull down the latest version of the library every time and just use that. Which is very difficult to then say well, what version were you using in this builds that had these problems?	
MN	OK, OK. Hmm.	
AC	Whereas sometimes, if you have a named or numbered build of a thing, but you know all of the problems with it, that can be more useful.	
MN	Hmm. No. OK.	
AC	But in in any case there's some risk analysis of those types of things.	
MN	Hmm.	
AC	And then before you get to any coding, there's still more reviews, verifications.	
MN	Hmm.	
AC	Then you get to the testing and obviously your testing is everything after your unit implementation.	
MN	Hmm.	
AC	So, you have your unit test, your subsystem integrations, your software, integrated system integrations, all of that is fine.	
MN	Umm. Hmm.	
AC	Then you have your validations, so you have all of the pieces.	
MN	OK. Hmm.	
AC	OK. But the last thing, the last couple of things are the final risk assessment, which is OK, we are looking for specific things as part of the risk assessment. So, hazards, failure modes, these kinds of things.	

MN	Hmm. Mm-hmm.	
AC	We are also doing a risk benefit analysis. No software or no medical device is essentially risk free. They all have some sort of risk about, yeah.	
MN	Yeah, that's correct. Yeah.	
AC	What is important is that the risks are sort of known and described, and that there's a benefit despite the risks of using the software.	
MN	Hmm.	
AC	So, for example, you know, our software will always have the risk associated with connectivity problems. If the risk of connectivity problems is unacceptable to the customer, they should not use our software in those environments.	
MN	Hmm.	
AC	So, for example, in an ICU environment where any loss of signal or connectivity would be intolerable because of the condition of the patient, essentially our software is not suitable in that environment.	
MN	Umm. Mm-hmm.	
AC	OK. But that's fine as long as you say that up front by your software. It's no problem. So, then you have these things and then you have final release and labelling.	
MN	Mm-hmm.	
AC	Which is OK like I talked about before, you know, you have a specific version thing at the end that you can then track when it's out in the world.	
MN	Hmm.	
AC	There's also I actual physical label obligations.	
MN	Hmm. OK, like the CE marking.	
AC	CE marking, yeah.	
MN	OK.	
AC	Whether or not, so it has this UDI, which is a new requirement, so that you can identify the device and the version and who made it and all that, you know that the manufacturer is identified on it, that there is a link to the IFU somewhere, these types of things.	
MN	Umm. Ok.	
AC	And then, they talk about regulatory release here on the technical release, you know.	
MN	Hmm. Hmm.	
AC	So yeah, that's just one part of meeting the requirements of 62304 within this thing.	
MN	OK, alright.	
AC	They have additional steps for AI and machine learning, and you also have these obligations for like keeping a list of known problems.	
MN	Mm-hmm.	
AC	But all of the other things that I have mentioned to you already, which are just outputs from your steps. Sometimes they will be	

	inputs to a step, like they are part of, and sometimes they just create documents to help you comply with the things that you said you would do.	
MN	Yeah. OK. Have you shared the link to this site in the chat?	
AC	So, I shared the link to this page here, the templates page, and then within the templates page you can see the various things, right.	
MN	OK. That's OK.	
AC	Yeah. So, ultimately the thing that how all of the things fit together essentially within all of the documents that you end up creating are this general safety and performance requirements checklist, right.	
MN	Mm-hmm.	
AC	So, this does not look great as a PDF. Because the tables are all squashed. Let me try and find a version of it. I'll try and find a version of it that I have.	
MN	Hmm. So, after you have done, your requirements, and then you have to check against this document, yeah.	
AC	So, the general safety and performance requirements are the general requirements of the Medical Device Regulation.	
MN	Hmm.	
AC	So, what this is saying is, can you see this tab now?	
MN	Yeah, yeah.	
AC	OK, so, this is our live QMS for example.	
MN	Mm-hmm.	
AC	So, we have a template document that we have numbered here that is this checklist, which is basically a restated version of the other one.	
MN	Hmm. Mm-hmm.	
AC	So, you have all of these general requirements and the way you meet these general requirements is by having in place the procedures that we talked about. For example, identify the known and foreseeable hazards with each device.	
MN	Hmm. Mm-hmm.	
AC	Alright, so you just point them to the compliance with the 14971 risk standard and how you implement that. The thing I wanted to just show was that, you know, so this covers all sorts of devices.	
MN	Umm. Mm-hmm. OK. Hmm.	
AC	So, you can see there's some devices have physical properties or by, you know, all that kind of things.	
MN	Yeah. Hmm.	
AC	So, the there's not too many when it actually comes to software, there's only four things that you have to do that are specific to software.	
MN	Mm-hmm.	
AC	And in that, it's basically just make the software and repeatable, reliable, and it has performance in line with their intended use.	
MN	Mm-hmm.	

AC	That's all it is. So that's where your defect reduction and identification thing comes in, right.	
MN	Hmm. Hmm.	
AC	So, you are trying to do that.	
MN	Umm. Umm yeah. Yeah. Hmm.	
AC	And ultimately it ends up in that V shape because of trying to do these things.	
MN	Hmm. Hmm.	
AC	So, you know, it's hard to have agile, fully agile processes that are repeatable. You know, sometime in the future you need to repeat those things. It's hard to do that.	
MN	OK.	
AC	Because you know, to be fully agile is to basically say, we don't document it in the same kind of straight way.	
MN	Hmm.	
AC	Whereas what you have done is you have kind of made that hybrid approach. So, for these four things what helps you comply with these things is this standard here (IEC62304).	
MN	Yeah. Umm.	
AC	And that has a bunch of things. So, for every standard that we would comply with, we would have a checklist against all of the clauses in that standard.	
MN	Hmm. OK.	
AC	It is very difficult to read these standards in that they are quite dense on information and yet they don't say a lot.	
MN	Hmm. Yeah.	
AC	But what's important is to, if you look at, say to this website as a resource is to look at.	
MN	Mm-hmm. Because it has, like synthesized the different standards.	
AC	Yeah. And to look at the mappings.	
MN	Ok. Hmm.	
AC	So, what this is going to say is in the standard, the clauses are towards like the second half or the end of the document and they will have clauses like this. And what you want to do is then be able to point to something that fulfils the obligations of those clauses.	
MN	OK, OK.	
AC	So, you can see here this standard has a clause to have a software development plan, right. So, the plan would simply be to, you know, do your framework, let's say, until you have good software, until you have a release and then you, you know, you would have some more detail in it than that.	
MN	Hmm, OK.	
AC	But essentially that's your plan for developing software.	
MN	Hmm.	
AC	So, you would have a document that just says that, and so on and so on. You can see that there are a bunch of things. This is just to say that this is the current approach for minimizing	

	defects in medical software. Your thing is kind of delving in detail into one piece of the bigger picture, and the only thing you're missing really, is how to integrate into the risk management process, your process.	
MN	OK. Mm-hmm.	
AC	Ok. So, hopefully this has been useful.	
MN	Yeah, very, very useful. Very useful.	
AC	Yeah.	
MN	I liked the way you have detailed the information. It has been really useful for me.	
AC	If this doesn't make any sense to you, you can send me an email or whatever and I will try and do my best to explain.	
MN	Yes. Yeah, that was my next requested to say I would really want you to continue your involvement in the project because the research is still ongoing. So, I would love if you can be open to my communication with you.	
AC	Yeah, no problem.	
MN	Yeah, because i have learnt a lot today and I see that you are the right person to contact.	
AC	Well, I don't know.	
MN	Yeah. Yeah. Yeah.	
AC	But you can see this software is useful because you can sign up for it for free and have a look around it.	
MN	OK.	
AC	So, this is just QMS after, but importantly within the QMS software, there are some introductory courses which are just videos on what this is for.	
MN	Umm OK.	
AC	It doesn't take long to get through this or anything. Some of it is about the product itself. So, you could skip that.	
MN	Hmm.	
AC	The wizard here is just. You know, talking indeed. You can see it's talking in detail about the different chapters of essentially the larger process of a medical company medical device company.	
MN	OK.	
AC	And you can see there is an episode there about a V-Model compliance developed, right?	
MN	Hmm.	
AC	So, everyone, almost everyone ends up in something like this simply because of the requirements of inputs to outputs. And the requirements of the standard you know.	
MN	OK. Umm.	
AC	You know, it's not too bad, but you know it's worth looking around at least.	
MN	Yeah, yeah.	
AC	So, some of this is for free, and you will be able to see kind of an outline of what a quality management system looks like.	
MN	Mm-hmm. OK.	

AC	And importantly, you'll be able to look at the most important thing is you can look at the templates that they provide.	
MN	Umm.	
AC	Which are just like, OK, this is how you might do it in a routine way.	
MN	Umm.	
AC	And you will see how they have written their different parts of the software development process.	
MN	Yeah. OK. Umm.	
AC	You know, like some of these things are not specific to the software development process but have implications for it obviously.	
MN	Umm. Yeah.	
AC	Yeah. Yeah, but maybe this pulls you away too much to kind of an industry side of things rather than a research side of things. I don't I know that it would be easy to quantify defect reduction in any you know, specific way.	
MN	Mm-hmm.	
AC	I don't. It's hard to measure it as a reduction if you know what I mean, but certainly all of the things that you have proposed fit in in the process.	
MN	OK. Alright. Yeah. So, uh, I think I have learned a lot from you and I appreciate your time. It has been very crucial and very important to me. I really don't take for granted for sparing your time to have this meeting. I really appreciate, and as I mentioned, just be open next time I reach out to you again. Your insights are really important. Yeah. Hmm.	
AC	OK, well good luck with all your research and your project.	
MN	Yeah. Thanks so much A.	
AC	OK. Take care.	
MN	OK.	
AC	We will chat again.	
MN	Thanks. Yeah. Yeah, we'll chat again.	
AC	Bye-bye. OK.	
MN	Yeah. So, I will come back to you with the transcription for any issues that may not have been captured. So, feel free for any request that I may have in terms of our discussion today, yeah.	
AC	OK. That's great.	
MN	Thanks. So, thanks so much.	
AC	OK, good luck with everything.	
MN	OK. Thanks.	
AC	Take care. Bye-bye.	
MN	OK. Bye-bye.	

E.5 Interview 5

Position: CEO

Experience in MDS: over 10 years

Interview date: 16 July 2024 10:30 to 11:30 am

Speaker	Transcript	Notes
MN	Hi EM.	
EM	Hey man, how are you?	
MN	I am good, how are you?	
EM	I am well.	
MN	I would like to really thank you for accepting to participate in this discussion.	
EM	Nice to meet you.	
MN	Nice to meet you too.	
EM	Sorry for missing on the last calls.	
MN	No, I understand, yeah. That's fine. But I really would like to thank you for accepting to participate, yeah.	
EM	Alright. Let's see how useful I am. It's really good group you are involved with in DkIT.	
MN	Yeah.	
EM	Just so nice companies spinning out from there as well. Pritty good ecosystem.	
MN	Yeah.	
EM	I am trying to think of the chap, what's the name of the chap who is kind of in charge of medical device regulation work there?	
MN	Professor FM.	
EM	Yes, FM. That's it. Yes. Yes.	
MN	Yeah. So, my name is Misheck Nyirenda. I am a researcher in the regulated software research center here at Dundalk Institute of Technology. Yeah. So, my research focuses on the use of agile practices to address defects in medical device software development. So, before you can also introduce yourself, may I have your permission to record the meeting?	
EM	Yeah. That's fine.	
MN	Ok. Alright. Thank you, E, for allowing me to record the meeting. So, you can now introduce yourself as well.	
EM	Thank you. My pleasure. So, my academic background is in computer science, specifically computational linguistics.	
MN	Umm. OK.	

EM	So, I would have studied areas around probabilistic parsing and machine translation, stuff like that. Back in the early 2000s. I then moved into the pharmaceutical sector and worked in designing and building large supply chain management ERP systems.	
MN	Hmm.	
EM	So, that got me started being involved in healthcare or life sciences and highly regulated sector as you know, and with a fair particularly in biotech, you also have a lot of the financial regulation and coming in as well.	
MN	Mm-hmm.	
EM	You had to start of Sarbanes Oxley coming in as well as the General pharma, Healthcare specific regulation.	
MN	OK.	
EM	My current company I co-founded with my brother and another chap called Scott Miller just over 10 years ago and it's in the area of patient feedback.	
MN	Umm. OK. Umm.	
EM	As in collecting feedback from patients on how they are doing on treatment and how satisfied they are with their treatment, and in health and social care services.	
MN	Mm-hmm.	
EM	The two types of feedback are generally called PROMS and PREMS, so PROMS stands for patient reported outcome measures and PREMS stands for patient reported experience measures.	
MN	OK.	
EM	So similar, but quite different in their objectives and the type of data collected.	
MN	Hmm. OK.	
EM	PROMS and PREMS have been collected for many many years in paper based forms and legacy IT systems, and then more recently, companies or health insurance companies and large healthcare systems, hospitals, etcetera, have been using off-the-shelf tools like Qualtrics and survey monkeys and other kind of general survey tools.	
MN	Hmm.	
EM	When they talk to us, it's generally at some sort of point in their enterprise architecture planning where their IT security team or somebody in IT is looking at their portfolio of IT systems and they are going, well SurveyMonkey or Qualtrics probably isn't a suitable tool to be using long term for collection of clinical	

	data, especially now we want to move on to more advanced data analytics.	
MN	Mm-hmm.	
EM	So, at that point, we generally end up being approached or part of a tender where they are looking for a domain specific or healthcare specific platform which adheres to the particular regulatory data compliance, data governance, data security requirements that they would want for a third-party system	
MN	Mm-hmm. OK.	
EM	So, this type of data collections has been around for many years in the pharmaceutical industry, in the area of clinical trials, and that's probably where it's most well-known.	
MN	Mm-hmm.	
EM	So, typically during phase three of clinical trials, you are collecting a lot of data and either directly as a pharma company, but more often as your clinical research organizations, your CROs who collect a lot of this type of data from patients during phase three of clinical trials, both your PROMS and your PREMS.	
MN	OK. Hmm. OK.	
EM	So, we are really taking that long established and well established type of software used in pharma or clinical research zeros and applying it within healthcare delivery and social care services.	
MN	Ok.	
EM	And what's driving the uptake of our software or this type of software more generally in health and social care in recent years has been regulation.	
MN	Hmm.	
EM	And the regulation changes slightly country to country, but it's generally one of two types of regulation in America. For Medicaid, for example, it's usually driven by the PROMS and PREMS data that's collected and is linked to some sort of financial reimbursement for healthcare providers or healthcare clinicians.	
MN	Hmm.	
EM	Or on the flip side, penalties if the PROMS and PREMS are demonstrating that these healthcare providers are not achieving good outcomes for the patient populations, that they are covering.	
MN	Hmm.	

EM	So, that's kind of a dollars and cents type of compliance requirement that's driving up tick.	
MN	Hmm. Ok.	
EM	And then the second would be more, we see it more in Scandinavia, where the health and social care systems there are mandating healthcare providers, hospitals, clinicians to collect PROMS and PREMS data during treatment and primarily because it's been demonstrated to improve treatment outcomes.	
MN	Hmm.	
EM	And it's kind of seen as the right thing and a good thing to do in terms of empowering patients, getting feedback from patients and influencing the direction of treatment.	
MN	Umm.	
EM	So, two quite distinct and different market drivers, but ultimately, we see ourselves as compliance software as in a lot of clinicians don't necessarily want to use our software. And similarly with a lot of end users of compliance software it's not sexy, it's not the first thing they want to use.	
MN	Hmm. OK.	
EM	But if it's mandated, they have to use it well, so be it.	
MN	Umm. Yeah.	
EM	So, that's a lot of information there in summary form, but I hope that's helpful.	
MN	Yeah, yeah.	
EM	And I kind of moved from introducing myself to talking about the company a little bit, but there you go.	
MN	Yeah, that's fine. So, uh, the format that I proposed in our discussion now is that I will ask a question and then you can share your views and the experiences where applicable. Yeah. So, do you want me to provide a brief explanation about the framework or you're happy for me to proceed to the questions?	
EM	Let's go ahead with the questions. Yeah.	
MN	OK, alright. Yes. So, let me just share the framework on the screen. I hope you are able to see the screen, yeah.	
EM	Uh, can you put it full screen?	
MN	OK, OK. Yeah.	
EM	There we go. Yeah.	
MN	OK. OK. Yeah. So, what I would like us to do is for you to share your views on the agile practices that are included in the framework in terms of addressing defects. So, starting with the requirements gathering stage, we have user	

	story, onsite customer/proxy and use case. So, these at this stage have been designed to complement each other, but the developers have the freedom or flexibility to choose a single practice or multiple practices to use at this stage. So, starting with the user story, what would be your views in terms of using it at this stage to address defects?	
EM	Yeah.	
MN	Talking about prevention side of things.	
EM	So, you are talking about proactively identifying defects, is it?	
MN	Yeah. Yeah.	
EM	Yeah, and well. We would maintain a risk register and we maybe quarterly review that and may identify some.	
MN	Mm-hmm.	
EM	Generally, we wouldn't pacify them as user stories in our system. But we wouldn't classify them as bugs either. What actually, no, they would be user stories. Yeah.	
MN	Mm-hmm.	
EM	So, we use Azure DevOps which was previously known as Visual Studio Online and to manage everything from requirements gathering through to continuous integration DevOps.	
MN	Ok.	
EM	And so, yeah, and when you set up, Azure visual studio as it was known, and which is now Azure DevOps, for the task management or development management part of it you would choose what framework and we do choose an agile framework as the methodology 2.0.	
MN	OK. Mm-hmm.	
EM	So, yeah, we would proactively identified defect as user stories, and we would record them in our customer ticketing system but assign them as tickets to ourselves.	
MN	OK. OK. So, what about use case? Using use case?	
EM	They would be captured as use cases within Azure DevOps. Yeah. Yeah.	
MN	Ok.	
EM	And there would be tagged as internal and the owner, so we would have a way to differentiate them from and user stories that come from requirements gathering or feature requests and stuff like that.	
MN	Umm, so do you find them effective in terms of preventing defects.	

EM	We wouldn't find the user stories effective as such, what we find the risk register and then the various monitoring tools that come with DevOps. Primarily Microsoft Defender, which identifies security risks and gives you a very comprehensive checklist and of items.	
MN	Mm-hmm. Ok.	
EM	It gives you a traffic light system of what's good in your system. What's amber? And then what's red that needs to be identified and based on this ongoing monitoring of the system and then we have Azure Application Insights which creates telemetry data from our application and would also give us quite a lot of information that ultimately may result in user stories being created and added to our sprints to be worked on.	
MN	Mm-hmm. Umm. OK.	
EM	So, the user story itself is just a way to capture that defect or potential bug, but if I'm understanding your question correctly, it's not I wouldn't see it as a useful tool to detect those defects or bugs or potentially defects or bugs.	
MN	Mm-hmm. OK. And onsite customer?	
EM	Since COVID, we have actually done very little on sites, which has been great, but I'm starting to miss it a bit and but really the same thing true or Support channels.	
MN	Mm-hmm.	
EM	So, our support email, our support chat and then with our larger customers we have weekly or monthly check in phone calls.	
MN	Umm.	
EM	So, certainly they would be key channels and particularly some of our power users or power customers would be very proactive in identifying current bugs or flagging potential bugs that maybe we need to investigate.	
MN	Ok.	
EM	And then certainly proactive and identifying maybe failings in terms of user experience that they would like us to address. But I classify them more as feature requests rather than bugs errors or defects.	
MN	OK, so, looking at this stage, have I missed any agile practice that you feel would be useful at this stage, the requirements gathering stage.	
EM	Well, yeah, this is interesting. So, the agile Framework itself, how does it broaden or how does it encompass the use of monitoring tools to proactively identify potential issues in the system.	
MN	Hmm.	

EM	So, as I said, customer feedback, be it onsite visits or through other channels, certainly is identified within agile documentation and training. But they use tools that generate a lot of telemetry data way beyond just simple logs. Application insights and takes all those logs and really presents it to you in a very… it increases the utility of log data really significantly because you have a really nice user interface for and alert system that will, you know, tell you based on the telemetry and log data and if things are starting to go a bit wrong.	
MN	Mm-hmm. Umm.	
EM	And we are beyond simple memory consumption. You know the kind of basic stuff that has always been in software applications. So, I think there's potentially a bit of a gap there in agile frameworks.	
MN	Mm-hmm. OK. So, moving to the software requirements analysis stage we have model storming, behaviour-driven development, pair programming, test-driven development, and onsite customer. So, I would like you to share your views on model storming at this stage in terms of preventing defects.	
EM	Yeah. I don't know much about model storming, to be honest, but we have gone down the BDD route, we have not fully implemented it. but out of what you've listed there, I would say BDD is our favourite approach and has the most promise as well.	
MN	OK. Umm.	
EM	And we're currently building a version 2 of our system and we have used a BDD alongside user requirements gathering as well.	
MN	Mm-hmm. OK. Do you find it useful in terms of preventing defects?	
EM	Yes, because they that. I'm trying to think of what kind of pseudo language that BDD uses but it's a very structured approach which is helpful and in generating lots of different scenarios.	
MN	OK. What about pair programming? what would be your views on this practice?	
EM	No. I would say it as a bit old fashioned at this stage pair programming. I wouldn't.	
MN	OK. And test-driven development?	
EM	Yeah, absolutely. So, that's a given and we wouldn't employ any developer who wasn't well versed in test-driven development and used that approach.	

MN	Mm-hmm. So, in terms of using it at this stage to address defects, to prevent defects.	
EM	Yeah. Yeah. So, they would really be writing the tests before writing the business code. And those tests are a key component or key part of our build and release pipelines in Azure DevOps.	
MN	Mm-hmm. Ok.	
EM	So, every time you build and release our software, whether it's to a preproduction or production environment, we have hundreds and hundreds of test scripts that are run.	
MN	Hmm. OK.	
EM	So, these tests are linked to the user story which is linked to the code that's deployed to production. So, both those new tests that are linked to the user story and then all the other hundreds of tests are all ran every time, which would form part of regression testing.	
MN	Umm. OK. And in terms of onsite customer or proxy at the software requirements analysis for preventing defects?	
EM	Yeah. So, wouldn't be onsite, but we would have, if I can say our proxies would be key customers in each of our key territories. So, they would be represented to us. They would be representative of the requirements and needs of everyone else.	
MN	Hmm.	
EM	So, you may have a different definition of proxy, but that's how I do understand it.	
MN	OK. Right. Any agile practice that you feel would also be useful for preventing defects at software requirements analysis.	
EM	Umm. I don't know. I am trying to think. So, generally the programmers who are reasonably doing a good job. Hence directly related to the business logic of the code they are writing, and the proxy customers would also be quite helpful. But the more serious and more impactful bugs, errors, and defects that I would be afraid of would generally be related to bugs, errors, and defects that may be linked to data leakage or compromising the security of the system.	
MN	Umm. OK.	
EM	They would come from a risk register and from Black Hat and white hat penetration security testing.	
MN	Mm-hmm.	
EM	I am not sure if either of those are identified within the agile framework.	
MN	Yeah. So, I think when you are looking at a requirement, you look at all those issues including security and the like. So,	

	moving to the SW Architectural Design, we have model storming, onsite customer/proxy, integration testing, and simple design. What would your views in terms of using model storming at this stage to prevent defects?	
EM	What's model storming? Is that brainstorming and coming up with lots of models and putting them into the backlog?	
MN	Yes. So, it called like just-in-time sort of practice where if someone doesn't clearly understand maybe a requirement or a design, then you can come up with a sketch to explain and then maybe they can clearly understand what that means.	
EM	Ok.	
MN	Yeah.	
EM	Well, we wouldn't probably do that because we know the type of software that needs to be built as in, we are building it to help our potential customers adhere to these emerging compliance regulations. So, it's not that we are building software that is completely the concept of the end users or customers.	
MN	OK.	
EM	So, that's perhaps why we would see model storming as applicable to us because we are kind of building to required industry standard, if you know what I mean.	
MN	Ok. Alright. What about onsite customer/proxy at the architectural design stage?	
EM	Again, more so for the user interface at the user experience, we would be looking at the web application and the app. Yeah but then on the business intelligence side of the things we have the data warehouse and the data visualisation of our solution. So, yeah, we wouldn't call it model storming, but we would ask people what they would like to see there and help them through that.	
MN	Mm-hmm. Ok. And integration testing?	
EM	Yeah. That kind of a core concept that anyone building this type of software would need in their development lifecycle.	
MN	Mm-hmm. Do you find it effective in terms of preventing defects?	
EM	Yes. Yeah. Yeah.	
MN	Ok. And simple design?	
EM	Yeah. What's simple design?	
MN	So, according to Agile, simple design means that you should make your design as simple as possible for easier modification and also easy understanding in terms of developers.	

EM	Oh Yeah. Ok. So, not just visual simple design, simple design in terms of documentation and requirements and so on, is that what you mean or?	
MN	Yeah.	
EM	Ok. And does it include user interface design?	
MN	Yeah, so all those to make sure that things are not complex that would hinder understanding.	
EM	Umm, yes, but simple design is inherent in first of all in the generally accepted documentation standards that most software development systems use.	
MN	Mm-hmm.	
EM	So, whether you are using Eclipse or you know, whether you are using an IDE and some third party software lifecycle management system or project management system, you know your documentation, your fields and data captured is generally the same across all different platforms. Or you know the structure for requirements document is pretty much the same. And in any country and anywhere I have seen, even across different sectors, so, agile or not agile whatever approach, those documents are pretty similar.	
MN	Umm.	
EM	More recently on user interface side of things, emergence of the WCAG (Web Content Accessibility Guidelines) standard, which the accessibility standard WCAG 2.0. That's generally a key requirements or mandatory requirement of public procurement of software systems now, particularly in Norway and Denmark.	
MN	Ok.	
EM	And the basic idea of WCAG is accessibility and simple design.	
MN	OK. Alright. Any practice or anything that you feel I have omitted on this stage, in terms of preventing defects.	
EM	Umm well yeah. Well, security-driven design really is pretty important.	
MN	Ok.	
EM	And we would definitely adhere to some of the accepted software security development frameworks. Primarily, OWASP, I am sure you have heard of them. So, OWASP popular top 10 vulnerabilities for different technology stacks**.** So, .net, Java, and so on. So, we would be very mindful of the top 10 OWASP vulnerabilities for .net core applications. I think the latest version is 2021 but we have been following OWASP since 2010.	
MN	Umm. OK. Mm-hmm.	

EM	And so that list certainly helps you proactively identify any bugs, errors, and defects and the security side of things.	
MN	OK umm.	
EM	So, basic one would be your XSS vulnerabilities. So, making sure fields can't accept weird characters that would allow hackers to amongst other things, preconditions and post conditions. So, I don't see that there. And more generally, penetration testing, black hat testing as soon as feasible.	
MN	Mm-Hmm. Ok. Now moving to the software detailed design stage, what would be your comment in terms of using model storming at this stage?	
EM	So, should I talk more generally or in the context of our own product and our own experience?	
MN	Yeah. So, you can talk about generally and then you can also provide your experiences in terms of your current process. Yeah.	
EM	Ok. Well, I mean, the onsite customer/proxy more generally would be the most powerful or most effective way of identifying bugs, errors and defects during the detailed design stage.	
MN	Mm-hmm. Ok.	
EM	And I would have thought that model storming is part of how you engage with those customers/proxies during that process, as opposed to as listed as, maybe I am not understanding correct. So, at the moment I don't get how they are present there at the same hierarchy.	
MN	Ok. So, at this stage we have model storming, onsite customer, integration testing and simple design. So, the idea here is that the developers would choose one practice, or two or all of them. So, if they choose all of them, the idea is that should start by applying model storming and onsite customer and integration testing. Once that is done, then they can follow that with simple design and onsite customer and integration testing.	
EM	Ok. But the model storming requires engagement with the customers/proxies outside of your organisation, would that be correct?	
MN	Yeah. That's ok.	
EM	So, model storming is predicated on meeting your customers onsite of meeting your proxies, would that be fair to say?	
MN	Yeah.	
EM	Can you do model storming without your customers or proxies?	
MN	You can model storm within the team in your organisation, yeah.	

EM	Umm.	
MN	For example, you have divided tasks among team members and then one is facing a problem. So, they would call another member to say I don't understand maybe this requirement, and then the other member would help by coming up with maybe a model which would be maybe a sketch and then explain to the member. From there the member would understand and then proceed to develop what is appropriate.	
EM	Yeah. Ok.	
MN	So, it can also happen with customers, you explain to them what they mean through a model so that you gain a clear understanding about what they mean about a requirement and what you would to provide as a system.	
EM	And then one other question. So, for the software requirements analysis, you have model storming and BDD. Again, I need to go back and recheck, but would it not be fair that BDD is closer to the actual code that is implemented than the model.	
MN	Yeah. So, BDD the way it is implemented is that, for example, you have a requirement, and then as a team you have testers, developers, architects and the like, you brainstorm about that and then each one gets an understanding. The testers would write the tests, the developers would now proceed to develop, and architects would now start doing the architectural design and the like. So, that's why it is included at the requirements analysis.	
EM	Yeah. So, BDD would have amongst other things, preconditions and postconditions that are formally captured. It goes back to the Eiffel object oriented Bertrand Meyer and his language Eiffel is based on that. So, it's pretty deep detailed. So, I am just surprised that you have model storming and BDD in your software requirements analysis, but then just model storming in the detailed design, when, in my opinion, perhaps BDD is more detailed and more closely aligned with the software detailed design. It's just an observation.	
MN	Mm-hmm. OK. OK. Yeah. Yeah, that's great. And that's why I am consulting so that I can get views on how it can be improved.	
EM	Hmm.	
MN	Yeah. So, this is a very important point that you have provided there.	
EM	Well, maybe. I don't know. We will see.	
MN	It is. It is, yeah.	
EM	OK. Let's keep going.	

MN	Yeah. So, moving to the software unit implementation we have pair programming, TDD, model storming, onsite customer and coding standards, and code ownership, as well as small release, and refactoring and unit testing and code review as well as the continuous integration.	
EM	Yeah.	
MN	So, I would like you to provide comments on this stage in terms of using these practices to address defects.	
EM	Yeah. Yeah. OK, so, Test-Driven Development is pretty much inherent in writing the code associated with any user stories that are generated in Azure or Azure DevOps system. So, whether those user stories come from customer tickets, or they come from proactively identified bugs, errors and defects.	
MN	Hmm.	
EM	So, following TDD would help us identify what those potential bugs, errors, the defects are, and to cover them.	
MN	OK.	
EM	And make a decision whether we want to cover them in the code or we want to just know when they happen.	
MN	Hmm. OK.	
EM	Uh coding standards. Uh, yeah. We would have our general coding standards that you would expect from good developers. So dry, don't repeat yourself and separation of concern. So, all the kind of main object-oriented coding standards you would expect.	
MN	Umm. Hmm.	
EM	And then fairly recently we have moved our software from being the monoliths really where the software code and the database and everything was kind of altogether, to a proper three-tiered system where we have an API which is separate from our user interface code which is separate from our database.	
MN	Umm. OK.	
EM	So, that means we can write user stories or deploy bug fixes et cetera to the API and deploy it and it doesn't change the code deployed on the API service or if we need to make an adjustment to the database or run a migration script or something or just running it against a database app service or the DB service and so on.	
MN	Hmm. Umm.	
EM	So, that architectural separation has been very helpful for us.	
MN	Hmm.	

EM	And again, just from a logical architecture point of view to be able to think about the three different parts of the system as three separate entities is very helpful in just thinking about them separately and identifying potential bugs, errors and defects.	
MN	Hmm. OK.	
EM	Unit testing absolutely is key. So, again, having those unit tests and other tests that run every time we run a build and release, whether it's preproduction or production is vital.	
MN	Hmm. Hmm.	
EM	Small releases. That's pretty accepted. Now most software companies will try and do lots of small releases, same as us.	
MN	Mm-hmm. OK.	
EM	And yeah, we have had to refactor our code quite a bit and we still have, and we have had a lot of technical debt maybe coming from previous early versions of the software where we worked with external contractors and there's been poor code.	
MN	Hmm.	
EM	As a result of that, we have spent many years managing and then coding out and refactoring some of that and work, which has been a pretty significant technical debt.	
MN	Hmm. OK. Hmm.	
EM	Umm, code review, we don't do it as a matter of course, but we would do it if we feel things are maybe going awry and we are getting a spike and maybe either bugs or anomalies.	
MN	Hmm. Umm.	
EM	Sorry, just give me one second.	
MN	Ok. Alright.	
EM	Hello.	
MN	Hi.	
EM	Sorry man. Sorry, I had the bank calling me, so I kinda had to answer that and I had to just step out.	
MN	No, it's OK. Yeah, it's, it's fine. It's fine, yeah.	
EM	So, I'm walking a little bit.	
MN	Uh ok. Alright.	
EM	OK, well, let's go ahead and see how we go.	
MN	OK, alright. So, you were talking on code review.	
EM	Oh yeah, I think, well, I think I was done with that. We generally only do code reviews if there is and if there is quality issues going on and we need a senior person to step in for a	

	while and start to do some code reviews, but we don't do all the time.	
MN	Mm-hmm. OK, then continuous integration?	
EM	Yeah, well, that's the basis of Azure Dev OPS.	
MN	Mm-hmm.	
EM	So, everything from your user requirements gathering to your build and release pipelines, continuous integration is done through it.	
MN	OK.	
EM	That's a fundamental part of how we work.	
MN	Hmm. Any practice that I might have omitted on this stage?	
EM	Well, I think to draw out the security-driven development a bit more and certainly I'd be looking to reference back to OWASP, which would be kind of more closely followed by the actual developers, but then at a slightly higher level, particularly in our field of medical device software, medical device development in general.	
MN	Mm-hmm. Ok.	
EM	And uh, some of the self-certification, some of the standards around data and security governance. So, cyber security essentials.	
MN	Mm-hmm.	
EM	Umm, what's the other one? Uh, Cyber security essentials. They are kind of the ones that lead up to ISO/IEC 270001 which is your full externally audited.	
MN	Mm-hmm. OK.	
EM	And finally, relation and cybersecurity essentials. What the hell is the other one? I can't think of it right now.	
MN	Umm. That's OK.	
EM	And then penetration testing. The use of automated tools or actual ethical hacking like black hat, white hat testing.	
MN	OK.	
EM	So, I think there definitely needs to be something in there that encapsulates that just beyond security driven development.	
MN	OK. Umm. OK.	
EM	And because really, it's always near the top of the conversation with IT security teams who are part of the procurement conversation when a hospital or a health system is deciding to buy our software.	

MN	Yeah, yeah.	
EM	You have the clinical people, you have the finance people looking, you know, negotiating on price and terms and conditions.	
MN	Hmm. Hmm.	
EM	But you also have the IT team who are looking at your certification for data governance and more often than not, well not that often, but the more sophisticated ones are asking the penetration testing report, fairly recent one.	
MN	Umm. Mm-hmm. Hmm, OK.	
EM	So, if a spin out or a company is using your tool in order to be prepared for the real world, so to speak, and hopefully selling, they need to be well prepared on data governance certification and they can start off with easy versions of self-certified certifications such as Cyber Essentials I mentioned and also to factor in the black hat and white hat penetration testing using automated tools and manual hacking as well.	
MN	Hmm. OK.	
EM	Because I would see if there's just security driven development in your framework, I would almost see it as just a token listing and I need to see it built out a bit more to be assured that, OK, that's been properly taken into consideration.	
MN	Mm-hmm. OK.	
EM	So, I won't say any more about it, that I've gone on and off about it.	
MN	Yeah. So, at which stage do you suggest to include the security-driven development or design?	
EM	For right from the start, completely from the start because your first version of your software will have an email feel or a free text field.	
MN	OK. Hmm.	
EM	And if you are looking at your top ten OWASP vulnerability list, #2 is preventing XSS, preventing cross-site scripting attacks.	
MN	Hmm. Hmm.	
EM	So, therefore your free text field needs to be character restricted.	
MN	Hmm.	
EM	So, right from the very start.	
MN	OK, alright. So, moving to the software unit testing, we have unit testing, onsite Customer or proxy, pair programming, and	

	refactoring. What are your views in terms of using these practices to address defects at this stage.	
EM	Yeah. Yeah. So, your unit testing, are you your basic most basic unit really of protection against any new bug service and defects.	
MN	Umm.	
EM	So, because they run, they provide the most code coverage and they run every production.	
MN	Mm-hmm.	
EM	Pair programming, we don't do it. I see it as a bit old fashioned.	
MN	OK. Mm-hmm.	
EM	And then, well, yeah, refactoring has been a big part for us because we have inherited a lot of technical debt from previous contractors and subcontractors that our current team has been dealing with, but I don't see that as an issue at a new product development.	
MN	OK. Hmm. Mm-hmm. OK.	
EM	At least it shouldn't be. You have got more fundamentally, quality issues going on if you are refactoring code that's only six months old.	
MN	Umm. Umm. OK. So, any other practice that you would suggested to include at this stage, software unit testing?	
EM	Yeah. I don't see stress or load testing there and that's pretty important.	
MN	Ok. Mm-hmm.	
EM	And so, for example, now we have, if you recall I mentioned, we separated our architecture into three different components, the API, the user interface could nor the front end code and the database.	
MN	Yeah. Yeah. Umm.	
EM	So, we were getting performance issues, and it was much easier for us to discover the root cause of some of them, because we could stress test our API service separate from our user interface separate from our database.	
MN	Hmm. Umm. Hmm.	
EM	So, that's stress testing identified and some entity framework code in the API. Which was and so entity framework dynamically generates SQL.	
MN	Mm-hmm.	
EM	But that SQL was running pretty was pretty poor SQL, so we threw the stress testing we identified that was the pinch point	

	and we just but rid of the entity Framework Code and rewrote those things as stored procedures and deployed in the database layer.	
MN	Umm. Hmm. Mm-hmm.	
EM	So, stress testing probably needs to be done from the start as well or stress test load test interchangeable.	
MN	Mm-hmm. Ok. And moving to the software integration and integration testing stage we have integration testing, pair programming, refactoring and unit testing.	
EM	Yeah.	
MN	Yeah. Can you share your opinion on these practices in terms of addressing defects at this stage?	
EM	So, for integration, integration testing, uh yeah, user acceptance testing. Yeah, well. To bring over. So, your simple design concept on your architectural design on the other side. Yeah. Umm. Well, it all depends on what your definition of user acceptance testing is, but I would say let's assume that you have usability or accessibility testing within your definition of UAT. So, I don't know if somewhere else in your documents you have it.	
MN	Mm-Hmm.	
EM	What's your definition of UAT or where are you taking it from? Is it defined within the actual framework?	
MN	Yeah, yeah, yeah.	
EM	OK. So, the observation I would make is when you reference back to your definition of UAT, does that definition include accessibility testing?	
MN	Mm-hmm. Umm. OK.	
EM	I suspect it may not. And if not, it should be included in there.	
MN	You are mentioning accessibility?	
EM	Yes	
MN	Ok.	
EM	Accessibility and the standard I mentioned previously, WCAG 2.0 is a requirement in lot of procurements.	
MN	Mm-hmm. Yeah, yeah, yeah, yeah.	
EM	Sorry, in an increasing number of procurements, request for proposals now, particularly in Norway and Sweden and little bit in Denmark.	
MN	Mm-hmm. OK.	

EM	So, that's for people who have colour blindness, people who have visual impairments, etcetera.	
MN	Mm-hmm. Hmm, OK.	
EM	So, if a municipality or communa in Norway is buying a piece of software for their citizens, IT healthcare or section there, they are now saying to you as a software or medical device vendor, this software must be fully accessible, and we measure that by… If you don't comply, we won't buy your software.	
MN	Ooh. Ok.	
EM	But you get my point, the definition of UAT has been around a long, long time, but I am not sure that your reference, your definition of UAT is one that includes accessibility testing at UAT stage.	
MN	Yeah, yeah. Hmm. OK. Yeah, I think I will incorporate this accessibility testing as part of the user acceptance testing.	
EM	Yeah. Well, perhaps you can. I can't answer that. It may be that it's fine for it to be part, defined under UAT. But you may find that it needs to be separate. I don't know.	
MN	OK. Yeah. OK. And refactoring at software integration and integration testing.	
EM	Yeah, that's all pretty standard stuff. So, I don't think there it is correct for it to be there. I don't have much more comments about that.	
MN	Mm-hmm. OK. Mm-hmm. OK. Alright. On software system testing we have user acceptance testing, onsite customer, pair programming, unit testing, and refactoring as well. What would be your comment in terms of using these to address defects at this stage.	
EM	So, would you see this as pre or post-release or both?	
MN	Yeah. So, this is the pre.	
EM	Pre.	
MN	Pre, yeah.	
EM	Hmm. That's fine, but perhaps some of the automated, some of the monitoring tools. At this stage you are pretty advanced in your life cycle.	
MN	Umm. OK.	
EM	So, you are at or nearly at the point of releasing your alpha or beta version of your software or device. So, you should have built up quite a bit. You have gone through user acceptance testing; you have a non-trivial number of people using the system. Therefore, the use of uh, say application insights within Azure DevOps and AWS has similar tools.	

MN	Mm-hmm. OK.	
EM	You have enough users that will be generating enough activity and logs and the system that warrants the use of anomaly detection systems.	
MN	Hmm, OK.	
EM	So, I think that's the right terminology.	
MN	Mm-hmm.	
EM	So, application insights within Azure DevOps, you deploy the script within your code which captures the telemetry data from your application, right. And is your application insights and the equivalent tools within AWS and other platforms will proactively detect anomalies in your system based on that telemetry data that it collects, which is generated by your user acceptance testing and your internal testing.	
MN	Umm. OK. Hmm, OK.	
EM	So, I think there is something you need to add there too.	
MN	Mm-hmm.	
EM	You could call this anomaly detection tools for the moment and maybe come up with a better term or more recognizable or industry accepted term, but that's certainly what we would be doing and have been doing at that stage.	
MN	OK. Mm-hmm. OK. Alright.	
EM	And something we are doing with we are adding a new prediction model to our software, which uses logistic progression. And we are definitely monitoring for anomaly using application insights.	
MN	Mm-hmm. Yeah. OK. OK. So, moving to the acceptance testing we have accept user acceptance testing pair programming onsite customer or proxy. What would be your opinion in terms of using these to address defects at this stage?	
EM	Well, yeah. So, pre-release I don't see a role for pair programming, but you probably have other people who are saying that they use it or it's relevant. And onsite customer or proxy, yeah.	
MN	Mm-hmm. OK. Hmm.	
EM	We have tried them and haven't had huge success, but I know they're getting better as you are kind of automated user tools.	
MN	Hmm. OK. Umm.	
EM	So, I don't know if you are familiar with selenium.	
MN	No.	
EM	Best tools. So, basically where you record using the system.	

MN	Mm-hmm.	
EM	And actually, this ties back to BDD.	
MN	OK.	
EM	So, with selenium you can actually. Oh, and I just the scripting language or the main scripting language used in BDD. It skips me but you can directly map it to selenium.	
MN	Umm. OK.	
EM	And you can essentially use your BDD repository to generate browser activity.	
MN	Hmm, OK.	
EM	And then that automated tool say like simple sign up form or a password reset form in an application. So, you would have BDD for user resets password that they go to, you know, company.x/reset, they type in their email address and then click the submit button. And then they receive an email, etcetera.	
MN	OK.	
EM	So, that's your BDD. So, you would BDD and then you would record yourself doing that, say in Chrome with the selenium automated testing tool and then you can run that test in every any browser you wish automatically. So, it's alongside your unit testing or your unit tests that run every time you do a build and release. All the automated user interaction with your system can also run.	
MN	Umm, OK.	
EM	Do you know what I mean. So, you can replicate a human doing user acceptance testing through using selenium or a tool like browser stack and it will replicate the human filling in those fields clicking a button on a page.	
MN	Umm.	
EM	Yeah, that's huge.	
MN	OK, so that's selenium.	
EM	Selenium, yeah. So, that probably comes under user acceptance testing.	
MN	OK. Is it a commercial tool or it’s an open source?	
EM	Selenium is open source.	
MN	Uh, OK.	
EM	And then you have commercial platforms that use the Selenium web driver engine.	
MN	Hmm. OK.	
EM	I think it's web driver is the open source part.	

MN	Hmm.	
EM	And now we've used it a little bit, not as much as I would like to have, but it's very widely used because otherwise you would need a team of testers testing across all different browsers, or to recruit different users and user acceptance testing. Some who have Chrome, some who have Edge but not that that matters not, or both, Safari etcetera.	
MN	OK. Umm.	
EM	So, UAT would be manual and automated and the automated UAT is important.	
MN	Mm-hmm. OK.	
EM	Yeah	
MN	So, it's important to have both manual and automated at this stage.	
EM	Yeah. Yeah.	
MN	OK, alright. And so just to be clear, what software development methodology do you follow currently?	
EM	We follow Agile 2.0.	
MN	Come again?	
EM	Well, it’s called Agile 2.0 within Azure DevOps if you can Google it.	
MN	OK. OK. So, DevOps.	
EM	Azure DevOps.	
MN	OK, alright. And any particular reasons why you choose to follow this particular methodology?	
EM	Well, because it's agile and at some stage we will and hopefully do IEC62304 certification which is, you know, in large part based on agile.	
MN	Hmm. Hmm.	
EM	So, that will hopefully reduce the workload in terms of well, we will hopefully be able to transpose our documentation around and how we do agile into the documentation required to achieve IEC 6304 certification.	
MN	OK. Umm. OK. Hmm.	
EM	Does that makes sense?	
MN	Yeah. Yeah. So, looking at your current methodology can be looking at this framework. Would you share your views only implementing this framework in your organization, specifically looking at addressing defects.	
EM	Could you repeat that?	

MN	OK. Could you share your views on using the Agile Defect Addressing Framework. This framework that we're looking at now in terms of addressing defects in your organization.	
EM	Well, I'm not gonna focus on the good parts and the parts I like because that's no good to you. But I will try and recap the parts that I feel we are doing that are missing, or maybe if they are not missing that they are not lost under something like UAT and so even though we don't do a huge amount, we do a certain amount of automated UAT, UAT testing using a Selenium.	
MN	Mm-hmm. OK.	
EM	So, I think that's important to feature automated user interface or frontend testing as part of UAT. Actually, as part of continuous integration, CI because you want to run those scripts every time you do a build and release the same as your unit tests and other types of tests.	
MN	OK. Hmm.	
EM	Secondly, then, and oh yeah, the accessibility stuff. So, we have covered that in a fair bit of detail, and I think that needs to be highlighted a little bit more particularly for software as a medical device because given its nature it needs to be highly accessible given the cohort end users who may have cognitive issues or visual impairment issues as well and just age etcetera.	
MN	Mm-Hmm. OK.	
EM	So, all of those factors I would say really needs to be at top level within your framework and not buried under some other heading.	
MN	OK. OK, alright.	
EM	And because pretty quickly, in the next few years, you are just not going to be able to sell to your customers unless you can show you adhere to WCAG 2.0.	
MN	Umm. Hmm.	
EM	And then thirdly, the two parts to number 3, one is, well, you have security-driven development. So, I would say an example of that is adhering to OWASP to help prevent vulnerabilities for whatever development framework you are using.	
MN	OK, OK. Mm-hmm.	
EM	And so, I would even go so far as to name check OWASP, either somewhere in your documents and then the other part to that is your penetration testing.	
MN	Mm-hmm. OK. OK.	
EM	So, your Black hat and White hat testing where you have human trying to break your system and your automated uh, your automated tools.	

MN	Yeah. Umm. OK.	
EM	Because you need that pen test report to even potential pilot. Well, you need it to primarily, for the context of this discussion, to identify bugs, errors, and defects, but then later on to reassure potential customers of your product, yeah.	
MN	Mm-hmm. OK. So, looking at the Agile-based Defect Addressing Framework, and then relating it to the standard IEC 62304, what would be your opinion?	
EM	Yeah. Umm. Well. Yeah. Can I add number 4 then?	
MN	Yeah, yeah, yeah.	
EM	And so just as we feel that jumping straight to ISO 270001 is too big a leap from where we are, and we have been around a few years we have made a decision to go with some of the self-certified data governance and security frameworks.	
MN	OK. Mm-hmm.	
EM	So, the cyber essentials being one.	
MN	OK.	
EM	So, I think perhaps there some certifications steps can be achieved by using your framework.	
MN	Mm-hmm.	
EM	That are part of the way to getting you to 62304. Oh CSA STAR, that's the other one CSTAR.	
MN	Come again on that one.	
EM	Its CSA STAR (Cloud Security Alliance (CSA) Security Trust Assurance and Risk (STAR)). It's similar to cyber essentials, it's a cyber security certification and the most basic version is a self-certified one where you complete a document saying we do these things and that allows you to self-certify and then the next step is when you become more mature to CSA STAR.	
MN	Mm-hmm. OK. Mm-hmm.	
EM	When you become more mature to do the next level, which is your externally audited version. So, I think for a lot of software companies who are using your framework.	
MN	OK.	
EM	So, I think for a lot of software companies who are using your framework, it's still maybe too much work or too big a jump to go from starting to use this to 62304 or 270001, and that perhaps a middle step or intermediate step is to look at your cyber essentials or CSA STAR self-certification.	
MN	Umm. OK.	

EM	So, my answer I think to your question is this, this helps get you to 62304, but I think in reality it may be helpful for a lot of particularly software companies to use this and then to work on getting data self-certification. Which one of those two standards I mentioned and then the next step after that is 62304 or 270001.	
MN	Mm-hmm. Ok. So, just to follow up on me this, do you think this framework aligns to the 62304?	
EM	Yeah, yeah.	
MN	OK. And then I think you have mentioned a number of things that you feel I have missed within the framework. So, that was my next question, but what are your thoughts on the adoption and implementation of this framework in your organization?	
EM	Yeah. It's good timing and I'm certainly would like to use it. Even looking at this, there's a few gaps that I know are there in our organization and this would be a good rubric to use.	
MN	Umm. OK. Mm-hmm.	
EM	Yeah, yeah.	
MN	Alright.	
EM	I do need to finish up.	
MN	Yeah.	
EM	I don't know if you are familiar with the post office scandal in the UK and you really should be watching it on YouTube. It's just fascinating because it's covering the horizon software system that Fujitsu built for the post office basically was so badly built and caused all these bugs, errors and defects that led to all these postmasters, 900 of them being prosecuted, and a lot of them jailed.	
MN	OK, OK. Umm.	
EM	You might have seen the documentary or the series a few months ago in January, about the post office scandal.	
MN	No, I haven't come across it.	
EM	You absolutely have to watch it. It's actually on YouTube now. They have to Fujitsu security engineer and a few couple of weeks ago they had the Fujitsu senior architect who architected the second version of Horizon.	
MN	OK. Umm. Umm.	
EM	It's going to be the case study in how not to build software and in the real world, consequences and impact on people's lives of badly designed software like it's just absolutely crazy what went on.	
MN	Hmm.	

EM	I have seen some of the code written on forums, so Fujitsu were being paid by the post office for lines of code, so that led to all sorts of perverse effects.	
MN	Umm.	
EM	So, for example, there was some requirements to a add a number and instead of just using an inbuilt double function, to reverse the sign.	
MN	Hmm.	
EM	It was to reverse the sign of a number. So, you could just multiply it by minus one, but instead what they done was they created local variables, multiply the number by two, subtracted it from the original number, and then substituted that.	
MN	Hmm. Hmm.	
EM	You see what I mean? They thought they would get the same results. But sometimes there was errors generated during that and there was no try catch block and that was directly leading to incorrect totals in the balance sheet of the postmasters which led the post office to prosecuting them because they were saying money was going missing, when in fact it was bugs, errors and defects coming from the horizon system. So, what you should look up post office on YouTube right now and watch the Fujitsu guy.	
MN	Hmm. OK. Yeah, I would definitely watch this one.	
EM	It's incredible. It's just incredible.	
MN	Yeah. Yeah. Hmm.	
EM	It's unbelievable. It's unreal. Yeah. Yeah.	
MN	Hmm.	
EM	Anyway, I need to jump in another call man.	
MN	OK, alright. So, the last question I was going to ask is looking at maybe a theoretical implementation of this framework in your organization. So, would you be open for me to do a theoretical implementation there.	
EM	Sure. Yeah, yeah, absolutely.	
MN	OK, OK. So, I will follow up with an email, so we can maybe arrange a date.	
EM	Yeah. Sure.	
MN	Yeah, you are based in Dublin, yeah?	
EM	Yeah, yeah.	
MN	OK. Yeah. So, I would really appreciate if that would be possible, yeah.	

EM	Sure, sure. Of course. Yeah. Give my regards to Fergal if you see him and I'll talk to you soon, alright man.	
MN	OK. Alright, definitely. Thanks so much.	
EM	Bye, bye.	
MN	Yeah. Bye.	

E.6 Interview 6

Position: Software Developer Technical Lead

Experience in MDS: over 7 years

Interview date: 24 July 2024 17:00 to 18:35 am

Speaker	Transcript	Notes
MM	Hello	
MN	Hi M	
MM	Hey, how are you?	
MN	I am good, how are you?	
MM	Good, good. Just jumped off another meeting.	
MN	Ah ok. Alright. Ok. Mm. Nice to meet you.	
MM	Nice to meet you. Can I pronounce your name	
MN	Ok. Misheck	
MM	Misheck.	
MN	Yes	
MM	Alright. Nice to meet you Misheck.	
MN	Yeah. So, first I have to thank you for accepting to participate in this research. I really appreciate. Yeah.	
MM	Alright.	
MN	So, my Name is Misheck Nyirenda. I am a researcher within the Regulated Software Research Center at Dundalk Institute of Technology. And specifically, I am focusing on using agile practices to address defects in medical device software development. Yeah. So, before you can also introduce yourself, may you allow me to record this discussion.	
MM	Sure.	
MN	OK.	
MM	Will this discussion be shared with anyone?	

MN	No. So, this is only for the purpose of my study. Not shared with anyone. Yeah.	
MM	Ok.	
MN	Yeah. So, thanks so much for allowing me to record. Yeah. Now you can also introduce yourself.	
MM	Uh, sure. So, I am MM. Well, but everyone calls me M. Uh, and I am a software developer technical lead now for some software developer. For the past nine years, 9 or 10. I did study in Ireland. I did study in Dublin Institute of Technology, which is now TUD.	
MN	Ok.	
MM	Umm and I am working currently with company F for the past nine years. Like technical lead in company F. Leading a feature team like a feature development team for company F.	
MN	Mm-hmm. OK.	
MM	So, yeah, I did live in Ireland for 12 years, I think 13 now, even more. Uh, yeah, so I came to Ireland for college and then stayed here for work as well. That's pretty much it.	
MN	OK. Alright. Thanks so much, Max. So, the format that I propose for our discussion is that I will ask you a question and then you will share your views and the experiences where possible.	
MM	Uh, sure.	
MN	Yeah. So, would you want me to give a brief explanation about the framework, or you would want me to proceed to the questions?	
MM	If you can give a brief explanation about the framework as well, that would be great. So, just to make sure I'm in line with your point of view as well.	
MN	OK, alright. Let me just share the screen.	
MM	I can see it.	
MN	Ok. You can see the screen, yeah?	
MM	Yeah.	
MN	Ok. OK. So, as I mentioned, my research focuses on addressing defects using agile practices. So, I have developed the agile-based defect addressing framework which you see on the screen. So, the practices that have been included in blue rectangles there have been designed so that developers can select either one practice or two practices or many or all of them at the particular stage, specifically to use them for addressing defects. So, for example, at the software requirements gathering stage, we have practices user story, onsite customer and use	

	case. So, they can use one practice, or they can use a combination. So, for example, they can use user story and onsite customer during requirements gathering. They can also use use case and onsite customer during requirements gathering. That is to prevent defects in terms of requirements. And where they have maybe decided to choose all practices at the particular stage, for example, at the software requirements analysis, there I am saying that for the practices included there, if you want to use all of them, then a best practice is provided, to say you should start with model storming and behaviour-driven development followed by pair programming, TDD and onsite customer. This is done so that you enhance the defect prevention at that stage. So, now as you can see on the left side of this framework, focus is the on preventing defects as shown in the green cogs there. And then on the right side, the focus is on identifying defects. I am saying that identifying defects because at this on the right side, you already have developed some code and therefore the focus is to see if there is any defect within the code. But the on the left side, you are preventing defects before you can proceed to the next stage. So, the framework also provides the flexibility to programmers to go back to a stage and correct something that maybe was missed at that particular stage. So, that's in brief about the framework. If you have a question, you can ask me.	
MM	No, it makes sense. Thank you for the explanation.	
MN	Ok. Alright. So, we proceed, and we start with the requirements gathering stage. So, I would like you to at least share your views on each practice that is included at a stage in terms of using them for addressing defects at that particular stage. So, we start with software requirements gathering stage, what is your view on using user story at this stage in terms of preventing defects?	
MM	So uh, usually the requirements gathering is done at the product level by product owners or scrum masters or product project managers because they are the ones that are in contact with the customers most of the time.	
MN	Umm OK.	
MM	Uh, but we are using user stories for that.	
MN	Umm.	
MM	And it's a combination of both user stories and onsite meetings with customers where we have like meetings with them and workshops to try to identify as much as possible.	
MN	Ok.	
MM	Uh, but you know sometimes it can be cases where we don't go really detailed, that the granular levels, and I guess that's when	

	defects can occur because sometimes people could make assumptions.	
MN	Mm-hmm. Umm.	
MM	So, I think assuming something needs to be stopped, right. It's better not to assume anything in at that stage. Think anything like questions need to be asked.	
MN	Mm-hmm. OK. Alright. So, if I get you clearly you have seen the benefit of using user story and onsite customer in terms of preventing defects at this stage.	
MM	Yeah. Yeah. We are usually doing that at very early, before we start the project with the customer.	
MN	OK. What about the use case?	
MM	Uh I am not sure. I don't think we are using that.	
MN	OK.	
MM	So, usually we have the user stories and onsite customers.	
MN	Hmm.	
MM	What would be the difference between the user story and the use case?	
MN	Yeah, in essence they collect the same information, but a use case go in detail in terms of the actions or the functionalities that are supposed to be included in the software. So, user stories more on the user part while use case you go in detail as a team of developers now to outline the actions that are supposed to be included within the software.	
MN	Umm. OK, alright.	
MM	But I am definitely sure about the user stories and the onsite meeting.	
MN	Mm-hmm. Alright. And is there any agile practice that I might have missed on this stage, that you feel would be important in preventing defects?	
MM	In the requirements gathering stage.	
MN	Yeah.	
MM	Uh I don't think so.	
MN	OK. Anything that you feel could be included within this stage that would help to prevent defects.	
MM	Umm. No.	
MN	OK. Uh, let's move to software requirements analysis and we have model storming, behaviour-driven development, pair programming, test-driven development and onsite customer or	

	proxy. So, I would like you to share your views on model storming in terms of preventing defects at this stage.	
MM	Uh. We usually, so, when the software requirements analysis, we don't go to the level of storming like model storming or behaviour-driven development. Umm, that's probably something that I would like to get to do.	
MN	OK. Mm-hmm.	
MM	Uh, but it it's not something that, as far as I am aware, it's not something that we are actively doing.	
MN	OK. And in terms of pair programming, are you able to use it here for maybe defects?	
MM	Uh. We try to minimize the amount of pair programming that we are doing because like we usually do have sessions where we're going through the code with for example, new developers or junior developers or developers that work in different areas and they have to do something in an area that my team controls.	
MN	Mm-hmm. OK.	
MM	Uh, but we don't really have pair programming sessions outside of that.	
MN	Hmm. OK, alright.	
MM	Uh, we do use TDD a lot so that during development and it's something that we enforce. And from what my experience, I think that's one of the best.	
MN	Mm-hmm. OK. Alright.	
MM	Yeah, because we are usually working together in the team to define the test cases and scenarios that we want to support. And then based on that, we will usually start writing the tests, the unit tests and integration test that we need to support and then write the actual implementation.	
MN	Hmm.	
MM	So, we mainly focus on test-driven development, uh, you know.	
MN	OK. Umm OK. Any involvement of the customer at this stage, the software requirements analysis stage?	
MM	We do sometimes have questions that have not been answered. So, at this level, we, like we do go the granular level, trying to identify like all the things that we need to implement and there are loads of cases where we do have questions that were not answered during the requirement gathering phase and we usually communicate that with our product owners and the product team, and they do have I think weekly or every two weeks. I think they have the meetings with the customer to answer concerning questions or gather more requirements.	

MN	OK. OK. So, in terms of the ones that you are using, TDD an onsite customer at this particular stage, have you found them to be effective in preventing defects?	
MM	Uh, yeah. TDD, test-driven development is very helpful in preventing defects. Because it helps developers not make assumptions.	
MN	Umm. OK.	
MM	And I think if everyone would practice that, it would be preventing defects a lot more. But unfortunately, even if you enforce test-driven development on a team, they are still loads of developers that would not use it.	
MN	Hmm.	
MM	And what we did find is that sometimes developers write tests after, right. So, they are not really using TDD, they are just trying to cover their implementation.	
MN	Hmm. So, what would be the main reason for developers not to first try write tests and then write them later?	
MM	Uh. inexperience, most of the time. I think sometimes not knowing the implementation details.	
MN	OK alright. Any other practice that you would like to be included at this stage to prevent defects?	
MM	No.	
MN	OK. And then we move to software architecture design stage. We have model storming, onsite customer or proxy, and integration testing, and simple design. Yeah. What would be your opinion on using model storming at this stage to prevent the defects?	
MM	Again, it's not something that we are using. At least from what I know, we are not really using model storming.	
MN	Ok.	
MM	We do use integration testing, so integration testing and designing the integration test is a big part as well of that. And I do feel that it's very good to prevent any defects.	
MN	Umm. OK. And any involvement of onsite customer or proxy at this stage.	
MM	I think like we do involve customers at all stages.	
MN	Hmm.	
MM	I'm trying to think at the level of integration testing. Umm. We would I think there would still be cases when we would have things like in terms of number of users or like if we are doing something about the load testing. So, I think those kinds of questions would be asked there.	

MN	OK.	
MM	So, yeah, like we do involve customers in those.	
MN	OK. So, aside from not using the model storming in your current process, do you think that model storming would be important to prevent defects at this stage?	
MM	Can you remind me what model storming exactly mean?	
MN	OK, so, model storming is a just-in-time practice whereby for example, you are in a team, and you have to divide the tasks amongst the team members and the team member has problems with their task so they can ask a fellow member to say can you try to explain to me. So, the one that has knowledge about the task would come up with maybe it's sketch to clearly explain the task to the member.	
MM	In that case, we're using this a lot.	
MN	Ah, OK, OK.	
MM	Sorry about that. I wasn't sure exactly what the model storming is, but like in that case, especially as a tech lead, I am involved in trying to give as much information as possible when I assign tasks to developers and usually, we do have questions or we don't know what, how, how something should be implemented.	
MN	Umm.	
MM	Umm I am the one that you know gets in a call with the developer and we are trying like we are actually going through step by step on what needs to be done.	
MN	Umm. Mm-hmm.	
MM	Uh, so in that case, yeah, it is something that we do use.	
MN	OK. Alright.	
MM	I am actually involved in that I think every single day.	
MN	OK. So, any other agile practice that you feel would be important to include at this stage, the software architectural design stage, specifically focusing on addressing defects or preventing defects in the architectural design.	
MM	So, at what step in this flow like is this the step where the development actually happens? The architectural design, like the developers are actually writing code.	
MN	Yes, so this mostly focuses on the architects looking at the architectural design of the software. So, much as these things can happen in parallel, but focusing on whereby the architects are now actually designing the architecture of the software. Yeah. So, do you have any other practice apart from these ones to say they could be included there?	

MM	We do have like our architecture, those like code scanning tools that would that usually help prevent defects like Sonar Cloud, for example.	
MN	Hmm. Mm-hmm.	
MM	Umm, so usually we do have our repository connected to this kind of tools like Sonar Cloud is one that would scan the code that is pushed, and it would look for the best practices and give feedback to developers.	
MN	Hmm.	
MM	And those usually, so, having that included in the architecture as a whole, would those give us good feedback about, you know, sometimes the amount of code that is covered by our tests or if we have a cognitive complexity that is too big or something like using good code practices to prevent defects. So, on an architectural level, I think that's important.	
MN	OK. Hmm. Alright.	
MM	So, I am not sure, probably it's part of the integration testing, but it it's different, right.	
MN	Hmm, OK. Moving to the software detailed design stage, we have model storming, onsite customer or proxy, integration testing, and simple design. What would be your views on using these practices to prevent defects at this stage?	
MM	So, again, we have the same 3 components, right, model storming, onsite customer, and integration testing.	
MN	Yeah.	
MM	So, this is the actual step where we are developing the actual features right?	
MN	Hmm.	
MM	Umm. Not sure if I. I don't think we have like. We are using Agile like the SAFe Agile framework, and we do have, you know, we work in sprints, and we do have refinement sessions where we usually go into the details of like usually product owners work together with the tech leads to define the tasks for the members and we do have refinement sessions where we go into details about how things should be implemented and how they should work.	
MN	Hmm. OK. Mm-hmm.	
MM	Umm. And we are using continuous development, continuous integration. Every feature that we implement has its own environment where we can test in isolation, and we do have a manual QA as well as automation.	
MN	Umm. OK. Umm.	

MM	So, all those tools are in place but most of the times we find that bugs or defects are coming in because something was missed during the design, right.	
MN	Hmm. OK.	
MM	Umm, so this level we are still involving customers, but it's not onsite. It is usually during meetings. It's usually through our product team that we solve all this question.	
MN	Hmm. Ok.	
MM	Umm. So, I am not sure if what we are doing is in any of those points.	
MN	Yeah, yeah, yeah. So, just to maybe to ask if integration testing or in this case we are saying integration testing, but actually we are saying that the integration testers should be involved at this stage. So, do you think that this would help to prevent defects at this stage?	
MM	Oh, yes. Yeah. Yeah. So, like what we have in company F is we have a team of like we usually have teams with few developers, we have a manual QA and automation engineer that is involved with the integration testing. And we do have quality refinements where we decide exactly what should be in the integration testing that is part of part of a specific feature. So, it is very important to have it at this stage yes and it is very important for preventing bugs because especially when it's not for the time when you are adding the actual features, when usually we won't pass development, or we won't say that something is complete if what we define is not there. But what we find is usually other people could be the same thing that are developing something new which is related to the same area, that could break stuff without realizing. And that's when integration testing is becoming very handy and giving new reports after like for every single pull request that we are doing is we are running the integration test, and you know exactly if something was broken by your changes or not.	
MN	Hmm.	
MM	So, yeah, I agree. It's very important to have it in this step.	
MN	OK. So, considering the complementary use of agile practices at this stage, what would be your views?	
MM	Uh, what do you mean, can you?	
MN	OK, like using the practices together. For example, combining onsite customer with integration testing and also where simple design with onsite customer and integration testing, using them like together complementary. Do you think they would help to prevent defects at this stage?	

MM	When you are saying on like. What would be the difference between a UAT environment and the cause? So, for example, we do have UAT user acceptance testing environments where we periodically post code in there or deploy the product into UAT environments and customers are involved in the development process.	
MN	Mm-hmm.	
MM	So, basically, we tell them every two weeks this is what we are going to do, deploy. This is what we are going to have. But we are not deploying anything until our integration testing are all like fully passed, right.	
MN	Mm-hmm.	
MM	And then at the same step after integration testing is fully done and everything you were happy with everything. Then we are pushing that UAT environment and that's when customers have access to the actual product and features, and they can test.	
MN	Umm. OK.	
MM	Right. So, in that case, combining them is very important. It's very important, especially when even after all the requirements analysis and gathering and all that, we do find a lot of time, a lot of the cases that there are misunderstandings between what the customer wants and what the product team and what the developers understand.	
MN	Umm.	
MM	So, having the customers involved at every step is important because you don't go too much into implementing something wrong before the user could realize it.	
MN	OK, alright.	
MM	The drawback on that is sometimes customers don't know what they're testing. So, sometimes, you know, product team needs to explain in more details about something. And then are cases where even though the user acceptance like the UAT environments are there, what we did find a lot of the time is that there was not a lot of customers actually testing everything properly.	
MN	Hmm.	
MM	So, having the customers being involved, actually involved it's important as well.	
MN	Ok.	
MM	So, what we did find is that some of our customers would have their own quality assurance team that would go through the things and that's the important even that step.	

MN	OK. So, any other agile practice that you would want to be included in the software detailed design stage to prevent defects?	
MM	No.	
MN	Ok. I think you haven't talked about simple design at this stage, would you share your opinion on it in terms of the preventing defects?	
MM	Yeah. So, what would you mean by the simple design on this step?	
MN	Yeah. So, in Agile, simple design just focuses on simplifying the design itself, not making it complex. And so that helps in understanding of the design itself by developers and also in terms of making changes to the design. If it is simple changes can easily be implemented by any other developer within the team. So, that is said that it helps to prevent defects. So, I would like to get your views on the on using it at this stage.	
MM	I am not sure if that works all the time. Because I know keeping things simple and I'm trying to always define everything at the granular level is ideal. But sometimes they are very complex things that need to be implemented.	
MN	Ok. Umm.	
MM	And yeah, we can't always keep it simple, unfortunately.	
MN	Ok.	
MM	Umm. But I do see the importance of it.	
MN	Alright. And let's move to the software unit implementation. So, we have practices included at this stage pair programming, test-driven development, model storming onsite customer or proxy, coding standards, code ownership, and small release, and refactoring and unit testing. So, may you share your views on these practices at this stage in terms of addressing defects?	
MM	Uh, yeah. So, I am trying to see, so this is something that I probably already mentioned. So, when the developer gets a specific task to implement in the product, uh, we usually uh create, like we are using Jira for example, for our tickets and we usually ask developers to create their own feature branch add like make the code changes in there. Umm. And then when that is done, we create pull requests and an environment like we specifically have environments created for that specific branch, for the feature branch before we can actually merge it into the developer branch, right.	
MN	Mm-hmm.	
MM	So, we do use all this, the continuous integration, so, when developers get the code, they usually know from the detailed design sessions that we had or the refinement sessions, what	

	needs to be done and usually have an idea of exactly the changes that they need to make in the application.	
MN	Mm-hmm.	
MM	Using TDD is enforced. Usually if the pull request, for example, doesn't have any unit tests, is not going to be even approved.	
MN	Umm. OK, OK.	
MM	We do enforce code reviews and approvals so at least 2 developers need to review the code and usually we find there are loads of cases where defects are even visible from the code review because we usually try to ask the people who know the code base better to do the code review.	
MN	Hmm. Mm-hmm. OK.	
MM	And even we do have Ownership of areas of the product, and usually if some features that we need to implement would touch other teams areas or something that other teams would own, we are normally asking other tech leads or other developers to do code reviews.	
MN	Mm-hmm. Umm.	
MM	And we do enforce approvals, so you need to have at least two other developers to approve your code. The manual QA to approve. So, usually the manual QA goes through the environment, and needs to go through the tests that were identified during the detailed design and if the QA is happy with the changes and they say that everything works correctly, right. So, they would approve. And then we also have the automation engineer or the person responsible the integration testing that they would also have to approve.	
MN	Umm.	
MM	But I think all these steps are, you know, taking a lot of time, but they are very useful for preventing the defects.	
MN	OK.	
MM	Uh, so, code ownership is one that I found it's important and coding standards as well.	
MN	OK. You haven't spoken about refactoring.	
MM	Yeah, we are using the SAFe agile. So, we usually work in PIs, so program increments uh and they contain 6 sprints.	
MN	Mm-hmm. OK.	
MM	OK. And every Sprint is for two weeks, right. So, usually PIs for us would last around three months.	
MN	Hmm.	

MM	Umm. And what we normally do is we do allow the teams to set up or allocate an entire Sprint for tech debts and refactoring. Or do it during the sprints, like a bit of refactoring every time.	
MN	Ok.	
MM	It is important because like tech debt can be added uh, when we don't have enough knowledge about an area or, so it is important to be addressed and what I do find useful is you have an allocated Sprint just for that.	
MN	Ok. Hmm-mm.	
MM	So, usually the team gets two weeks to fix defects and do refactoring on things that needs to refactoring.	
MN	Hmm.	
MM	Usually when we implement the features, we would find places that we know they need refactoring. Uh, we might not, like if it's small refactoring, that can be addressed within the scope of that ticket. Uh, we do advise them to deal with, or sometimes refactoring could take longer and we do try to make sure that we have enough code coverage and in our integration testing before can actually do refactoring on some cases.	
MN	Mm-hmm. OK.	
MM	Yeah, it is important.	
MN	Alright. And the small release?	
MM	A small release. Yeah, I think so. So, like a small release is that for customer? Because it depends on what we mean.	
MN	Yeah. So, it could be for a customer, or it could be for maybe a higher ranking team to say we have finished this piece, can you check it. So, it would be a small release of a functionality within the organisation.	
MM	Yeah. So, that's what we are trying to achieve with the feature environments. So, every feature has its own environment, so that would be considered a small internal release and we only we only put that into the product after a team is happy with the functionality and then at the end of Sprint, usually we are not doing that every Sprint, but I think at the end of every three sprints, we do have what we call an end-to-end team that performs manual QA.	
MN	Umm. Hmm.	
MM	So, that team is specifically only testing the end-to-end problem. Like every three sprints we are building an environment specifically for that, and that's when, uh, that team is checking the things, like they are doing sanity testing and performing the task to make sure that you know the release is good.	

MN	OK. All right.	
MM	So, what I found useful like for this small release is they are useful to have them internally. So, we do have like a nightly we call it a nightly release.	
MN	Umm. OK.	
MM	So, every night we are building an environment that the entire company development team can use and a lot of the times we do find that that environment you know, can catch defects by, like maybe one team is doing something that another team doesn't have enough code coverage or and they could break other stuff.	
MN	Mm-hmm. Umm.	
MM	So, it's important to have that. So, it depends on what the small release is.	
MN	OK.	
MM	They are important to have internally, I think, yeah.	
MN	Ok.	
MM	And then to the UAT environments, umm, I am not sure exactly how often they get a release, but I think usually the aim is for every Sprint.	
MN	Mm-hmm. Hmm. OK. Any agile practice that you think could also be included at this stage? For preventing defects or addressing defects.	
MM	Yeah. One thing that I find useful, and I am not seeing it here. So, every Sprint we have retrospective sessions where we discuss about the Sprint and how the Sprint went and every person in the team has the opportunity to say if something went wrong or what they learn in that Sprint. And usually when we have, because you know defects, they always happen, right? So, usually when we do have a defect, we are trying to come up with the reasons why it happened and how we can prevent it in the future, right?	
MN	Umm. Umm, OK, so that is discussed within the retrospective?	
MM	That is discussed during the retrospective meetings, yes.	
MN	OK. And do you find it to be very effective in terms of addressing defects within your product.	
MM	They are effective, yes, they are. Yeah, because usually we have actions. So, usually from the retrospective meetings we have actions that we need to take in order to better refine the tickets so next time we are touching a similar area, or we do something, we learn from the mistakes, and we try to put them in the next for refinement sessions.	

MN	Hmm. Umm. OK.	
MM	So, yeah, I think the if the retrospective meetings are done well, they can be very helpful.	
MN	Umm OK. Alright. Any other item that you would want to be included at this stage?	
MM	I did have a point about knowledge gap, like sharing knowledge with other people in the team. I thinks that's probably included in the pair programming.	
MN	Yeah, and TDD as well.	
MM	Ok.	
MN	Hmm. OK, so let's move to software unit testing. So, we have unit testing, onsite customer or proxy, pair programming, refactoring at this stage. So, would you comment on these practices in terms of addressing defects at this stage?	
MM	So, I am trying to understand what this step is exactly cause so the unit implementation if used in relation with TDD we would already have the unit testing. Umm, so, is this extra unit test on top of TDD.	
MN	Yes. So, the thinking is that you have maybe shared tasks within your organization and each maybe for example, let's say you have three teams and within the team they have their own functionality to develop. So, when they are developing that they can use these practices and then after completing developing the functionality, they can now take it for unit testing where someone now would use the unit test that they had already written to verify that what has been developed is a correct in terms of the functionality that was defined. So, you can unit test within the development team itself, and once the functionality is completed then someone else would verify the unit itself. And that would now go to integration where the different functionalities or units are now integrated together to form the complete product. So, that's the other step that we are going to.	
MM	OK. So, like we do, you have. So, usually we don't say that a feature is complete until we have enough unit tests to cover it.	
MN	Mm-hmm. OK.	
MM	And then, like every team is responsible for writing the unit tests and the automation tests.	
MN	Hmm.	
MM	So, we do have unit tests which is like developers writing their own test to test the units that they are writing and then we do have automation tests where the automation engineers are writing tucking style tests you know, with like using the user	

	stories. So, those would come in the later. So, I guess that those will probably be the integration tests on the fly.	
MN	Yeah.	
MM	Uh, but the outside of we don't really use unit testing cross teams. We do have all the unit testing for the project because we have a big project and let's say 5 or 10 teams working together on the same project and we do have a common runtime for the unit tests that we run and usually contain the unit test for all the teams.	
MN	Umm. Hmm. OK.	
MM	Alright. So, after we develop something and when we are happy to create a pull request, we do have pipelines that do check the entire unit tests for the entire product to ensure nothing is broken.	
MN	Hmm Umm. OK.	
MM	Umm, so unit testing is important but I'm not sure how useful in my point of view, would be to do the cross team unit testing.	
MN	Hmm. OK.	
MM	Because like if I would have to review or write tests for another area of the application which I'm not familiar with, would probably be very hard for me to do that.	
MN	Umm. Umm. OK. Alright. Uh let’s proceed to the software integration and integration testing. So, we have user acceptance testing, onsite customer or proxy, integration testing, pair programming, refactoring, and unit testing, as well as the continuous integration. Would you share your views and the experiences on these?	
MM	So, one of the problems with this is that the integration testing project or it can take a really, really long time to execute. So, depending on how big a project is, this could take a few hours.	
MN	Umm.	
MM	And what we did find is that we had to break the integration testing into kind of two areas; one we call the smoke test, to ensure that the functionality and like the basic functionality for the customer works fine. Uh, so in the smoke test, we only have a subset of tests, we are testing the most important aspects of the product. We are trying to minimize the amount of tests that we are putting there, just so that it doesn't take too long to run.	
MN	Mm-hmm. OK.	
MM	Especially if you have, like we do have loads of pipelines for every single feature and it is important to have those environments creating in fast, like quite fast. Because otherwise they are frustrating, and you need to wait. So, sometimes it is a	

	problem because those smoke tests would not cover everything, and then the full suite of integration test would be run probably nightly, like every night when we create the small release, the nightly release.	
MN	Mm-hmm.	
MM	And if there would be a way to improve the smoke test, like the small performance on that because we do find sometimes that we find we find out too late about some defects that were not captured in the smoke test suite, but they are captured in the nightly and that that one times can cause delays.	
MN	Mm-hmm. Mm-hmm.	
MM	We are usually trying to put it the environment that we build before we release it to a UAT environment. For example, we would ensure that the full system test will run, but you know speed is usually a concern.	
MN	Umm. Hmm. So, any use of pair programming at this stage?	
MM	Yes. When automation engineers write the tests, we usually do have sessions with them to explain some of the things or help them with the features.	
MN	Mm-hmm, Ok.	
MM	So, again, it's something to speed up the process of writing automation tests and making them reliant.	
MN	OK.	
MM	So yeah, it is. We only recently started to do that, and we did find out that the quality of our product did improve.	
MN	Mm-hmm. No. OK.	
MM	Uh, just because again, assumptions is the worst enemy, and the automation engineer would sometimes assume things.	
MN	Yeah, OK	
MM	And yeah, pair programming between automation engineers and developers is working.	
MN	Hmm. OK. And any refactoring?	
MM	Refactoring at this stage I think is mostly to do with performance. So, like we do have a time limit on some automation tests. For example, we have endpoints that we create for specific like from the back end point of view that would execute some business logic, and we do usually put like testing in place to ensure that it doesn't take too long to execute a specific process so that performance is good.	
MN	Hmm.	

MM	So, usually if performance is affected, we do go back to the step and we need to send back the code to be refactored.	
MN	OK, umm. And continuous integration.	
MM	Yeah, continuous integration is, yeah, it like we have the pipelines. So, they are the pipelines that run, so make sure everything builds, all the unit tests passed, all the automation test pass, deployment is done successfully into a brand new environment.	
MN	Mm-hmm.	
MM	So yeah, the continuous integration is very important. Happy step by day.	
MN	OK. Then onsite customer or proxy at this stage.	
MM	The onsite like the user acceptance tests are usually done in our company at the end of the Sprint, so we don't do it for every team.	
MN	OK. Hmm.	
MM	So, because we work in sprints, we usually put like all the teams will develop something like we have a plan to develop something and then into a Sprint and then at the end of the Sprint, we would deploy the full like from all the teams. All the team will deploy that to the user acceptance testing environment where the customer can actually see the progress? But yeah, I think it is important there.	
MN	OK. So, the involvement of customers here, you are saying if I get you correctly is through the user acceptance testing?	
MM	Yeah.	
MN	OK. And then any other practice or any item that you feel could be useful in terms of addressing defects at this stage?	
MM	Uh. I am not sure where to put it, but the one thing that I did find is sometimes, when the user acceptance testing happens, they usually happen after the teams have developed something in the previous Sprint, right? So, you are developing something now, but it will be tested, like the customers are actually testing something that you developed, you know, one week ago, right? And then in the meantime, the development team are actually focused on new functionality.	
MN	Hmm.	
MM	So, I think it needs to be the priority that is given to defects when they are found versus the new functionality. They need to be carefully planned.	
MN	OK.	

MM	So, sometimes you know, it's hard to say, fix a defect versus develop something new for the customer. So, giving a priority or a criticality to the defect is very important because sometimes they could, you know, even if the user or the QA wouldn't tell you that no, there is a problem if it's not addressed properly, and it's not planned carefully I think it would affect. So, planning or having someone to give a priority of defect versus a new feature it would be important to prevent even further defects.	
MN	Hmm. OK. So, you are recommending prioritizing defects at this stage, instead of moving forward to developing a new feature, right?	
MM	Yeah.	
MN	OK. And moving to the software system testing stage, we have pair programming, unit testing, refactoring, user acceptance testing and onsite custom or proxy. I would like you to share your views on these practices at this stage in terms of addressing defects.	
MM	So, I think this would be the same as what I said previously, right? Umm so software system testing.	
MN	Yeah. So, this is where now you have a complete product. You have integrated all the functionalities, and you have a complete product. So, you are assessing to say is it achieving what was stated in the requirement.	
MM	OK. And what would be the pair programming and unit testing at this stage or refactoring?	
MN	Yeah. So, the idea of including pair programming there is that in case there are minor issues that would be sorted out at this particular stage, you can do them there, but if they are bigger issues you need to go back to a particular stage where you need to rectify that. So, if the case is that you have a minor problem there, for example, maybe the requirement was that a button should be maybe green, and then it is not showing green, so that that can easily be rectified within that stage and then once you do that you need to check to say you have not broken other things. So, that's why I including pair programming there.	
MM	OK. Yeah, we would usually just go back to software detailed design. So, every change, we are trying, not like we are, we are not doing any patches. So, we would usually like if a defect is found, we would create a new development task to fix that defect, go through every stage, from the software implementation, write unit tests for it, make sure the coding standards and everything that is involved at that step.	
MN	OK.	

MM	So, if I understand that correctly. That would be too late, right? Unless it's probably unless you refer to the same thing. So, it will probably go back, and fix those and ensure everything is working because yeah there is a very high chance that something is going to be broken.	
MN	Mm-hmm. OK. Hmm.	
MM	So, I feel that for me it would be too late to do any changes at that level. So, it would have to go back to the software detailed design.	
MN	OK. Alright. And so that applies to the refactoring as well, yeah.	
MM	Yeah.	
MN	OK. Umm. Any practice or any item that you fee could be usable or could be useful at this stage in terms of addressing defects.	
MM	Umm. I think ownership of those defects would be important to know. Uh, like if we do have code ownership or feature ownership for the product level, uh would be usually important to send it back to the exact team that would have the knowledge.	
MN	Mm-hmm. OK. So, uh, just to be clear, you are suggesting that code ownership could be used here in terms of having the personnel that maybe are responsible for a particular feature to be present here. So that once a defect is identified, they should be able to know and then take a responsibility for that and rectify it.	
MM	Not the person specifically, but like someone for example, like the development teams would have a product owner, the person that would be responsible for the product development, like from the product team, and they would know exactly how or why something could happen, right.	
MN	Umm. Mm-hmm. OK.	
MM	So, having them give their views on that, probably.	
MN	Mm-hmm. OK. Alright. Let’s move to the acceptance testing stage. There we have user acceptance testing pair programming, onsite customer or proxy. May you share your views and experiences, if possible, in terms of addressing defects at this stage.	
MM	Umm. So, that would be again part of the user acceptance testing, right. Yeah, I am not sure what extra to say at this step.	
MN	Umm. Umm.	

MM	So, what would be the difference between the software system testing and the acceptance testing cause they both have the user acceptance testing in there?	
MN	Yeah. So, at the software testing we are saying that you can have user acceptance testing there. These could be internal people like proxies and then they would provide feedback in terms of defects there and then acceptance testing is whereby now these are the actual clients that you developed the system for, so here they will be checking…	
MM	OK, so this would be like a pilot for example.	
MN	Yeah, yeah, yeah.	
MM	OK, OK, so this is the real time.	
MN	Umm.	
MM	This is the real. So, yeah, we would usually. So, if UAT is successful, right, and we would usually get timelines to say, OK, if the customers are happy with the UAT environment, the next step before giving this so, right, would be to have a pilot customer or pilot store where we do actually get real data and real customers, and they would perform those tests on the real environment.	
MN	Mm-hmm.	
MM	And a lot of the times, you know, some defects that nobody thought about could be found in that stage.	
MN	Hmm.	
MM	Uh. So yeah, we do. It is important to have that level of testing before it can go out to everyone.	
MN	OK. Mm-hmm. OK. All right. So, we have discussed I think a number of practices within the different stages. So, my next question is looking at this framework in general now, what would you suggest as improvement in terms of addressing defects in the medical device software development?	
MM	So, one thing that I didn't notice before is that you do have from the left side at every single step you do have an arrow that goes back to the development set, right?	
MN	Yeah.	
MM	I think that was my confusion about the step. So, even like for example, if you find something at the acceptance test, the last level, so you go back to requirements gathering and do all the steps, right?	
MN	Yeah. So, depending on the defect. Yeah. So, at the acceptance test there, I think the best option is to go back to the requirements because here it means it's the real customer telling you that what you have done is not correct. So, it means	

	you need to go back. So, the idea with this framework is that as much as possible, when you get to the acceptance testing, you have cleared most of the defects and there is little chance of the user refusing your product. So, that's why we are saying that as early as possible, you need to implement proactive measures to prevent defects. So, starting with requirements gathering, then requirements analysis, make sure that all those are captured properly. That's why we are including there to say you need to involve proxies or customers, so you are in agreement throughout. Yeah. But if anything happens at any stage requiring that it needs to be corrected at a particular stage, you need to be agile and do that, to correct it, yeah.	
MM	Yeah, yeah, yeah. OK. That makes sense. So, even on the software unit testing that you have there, if you find a defect or so that needs to go back to the detailed design and implementation, make sure that you correct it.	
MN	Yeah, yeah. Yeah.	
MM	Ok. Yeah. So, it makes sense, yeah.	
MN	Umm.	
MM	Yeah.	
MN	OK. So, any suggestions for improvements?	
MM	Umm. I think the only one that I had which was about the balancing between the fixing of defect and working on the new functionalities that I wasn't sure where it's included here. But no other than that, I think it kind of includes a lot.	
MN	OK. So, looking at the current process within your organization and looking at this framework, what would be your opinion in terms of implementing it within your organization, do you think it would help to address defects?	
MM	I feel that we are already using a lot of this. Uh, so, I think the only one that we don't always have is the like we don't have the onsite customers or proxy at every single step. They are usually involved at the requirements gathering and the software requirements analysis and then the system testing and acceptance testing.	
MN	Mm-hmm.	
MM	So, they are involved in the first two steps on the left and the last two steps in the right.	
MN	Umm. Mm-hmm.	
MM	So, uh, but I am not sure how would you involve a customer at every single step. Would that mean You will have a customer to test everything single, like will we need to have a person in every single development team, a customer.	

MN	Yeah. So, this could be a real customer or a proxy. A proxy could be someone within your organization who would provide the maybe important feedback on whatever is happening there.	
MM	That would be the quality insurance assurance engineer.	
MN	Yeah, yeah.	
MM	OK, so like a manual QA that we have. OK	
MN	Hmm.	
MM	I do feel like we do already implement. We do already have a lot of this, and it is very useful for addressing defects. Umm.	
MN	OK. Mm-hmm. OK.	
MM	But one thing that is not done pretty well is, uh, they are not always in Sync. So, a lot of developers would. So, we are in for like we are enforcing TDD, but sometimes not all the areas like you wouldn't be able to catch all the areas that don't have uh or the developers that don't use TDD or in terms of unit testing, sometimes you wouldn't have everything unit tested, and defects would still appear.	
MN	Mm-hmm. OK.	
MM	But the process itself, it looks very similar to what we do in the organization already.	
MN	Ok.	
MM	And I think it is very useful because it we did, uh, so in my time in this organization, we did use a lot of different methodologies, right. Like we did you waterfall, for example.	
MN	Hmm. Ok.	
MM	And then it took a long time for us to get to a stage where we are close to this. Uh, and it is very important and useful.	
MN	Hmm. OK. So, the next question would be that would you implement this framework in your organization?	
MM	Yeah, I am not sure if I have the power to choose that.	
MN	Yeah, but looking at the framework itself and the benefit that may be it may offer, would you recommend using it within your organization?	
MM	Yeah. Like as I said, I think I feel that we are already using it, so I would recommend it to anyone because we are using most of this already.	
MN	Mm-hmm. Mm-hmm. OK.	
MM	So, we do have continuous integration pipelines, feature environments, unit test at every single step, feature environment for every single developer that does something. Umm so, I do feel that we are doing most of this already.	

MN	Mm-hmm. Alright.	
MM	So, and it does work for us, right. So, it's definitely useful.	
MN	Hmm. OK, alright. So, the other question would be looking at this framework and also considering the standard IEC 62304. Do you think that this framework conforms to that standard?	
MM	Can you repeat the standard?	
MN	The IEC 62304.	
MM	And what's that standard?	
MN	That's the standard for medical device software development life cycle.	
MM	Yeah, I am not very familiar with that standard.	
MN	OK, OK, alright. Ok, so, any other suggestions that you feel could be important in this framework?	
MM	Umm. I think it's just the uh customer and stakeholders communication. But I think you do have this really included.	
MN	Mm-hmm.	
MM	Uh, like it is again I put an emphasis on the importance of prioritizing defects when they come versus the new feature development, and I think the way the communication between customers and stakeholders, or product team is very important.	
MN	OK. Umm. OK. Alright. So, going back to our discussion, I think at some point when I was asking a question about software unit testing, and I think if I got you correctly you said that this particular stage is part of the software unit implementation. Is that correct? In your organization.	
MM	Yes.	
MN	OK, so you combine the software unit implementation and software unit testing as one process.	
MM	Yeah.	
MN	OK, alright.	
MM	So, usually the developers responsible for implementing something are also responsible for writing unit tests to cover that area, but it is also part of the TDD, right.	
MN	Mm-hmm. Mm-hmm. OK.	
MM	So, in order to implement something, you need to have unit tests to.	
MN	Mm-hmm. OK.	
MM	So, we usually start with writing unit tests that fail. write enough codes to make sure that the unit tests passed and then you have your unit testing, and we don't always just stop with	

	the tests defined at the TDD step right. So, usually developers are encouraged to expand on the unit test to make sure that they have enough coverage.	
MN	Hmm. OK, alright. And so just to again to ask, you said that the current software development methodology are following is.	
MM	Is SAFe.	
MN	OK. You are using a SAFe.	
MM	Yeah.	
MN	OK so. Is it that you are like subscribing to the organisation that created it or you have a license? I want to understand that.	
MM	Uh, yeah, I think we do. I am not involved or don't know about it, but I do know that we did have multiple times somebody that developed the same methodologies in the organization, giving us workshops on how to how to implement SAFe methodologies.	
MN	Hmm. Mm-hmm. OK, OK.	
MM	Umm, he was usually involved in a lot of our like we even have. So, we do have product increments that last like they say for like three or four months.	
MN	Mm-hmm.	
MM	So, 6 sprints and we do have people from the organisation involved in. Uh, so I am sure we have the licenses.	
MN	OK. Alright.	
MM	I am trying to remember who, what his name was.	
MN	OK. OK. So, the other question was for me to ask if, I know that you already use this. You are already Agile, but the other question was going to be to say would there be an opportunity of maybe having a desk-based implementation of this framework in your organization?	
MM	Uh desk-based.	
MN	Yeah, a desk-based to say maybe from the previous project that you did. And maybe looking at for example you said that you used a number of frameworks or methodologies like waterfall. So, from the projects that maybe you implemented waterfall or other methodologies and then need to go through that on a desk-based to say OK if we had used this framework maybe would have reduced defects or would have maybe released quicker or what.	
MM	Yeah.	
MN	So, this is not like you have a new project and then you are following this, but just working through, sit down with someone like yourself, I can sit down with you and then you	

	look at the previous project that you did and then we walk through to say OK if we had the used this one this would be maybe the outcome and the like.	
MM	Yeah. I am not sure because we only recently moved all the teams to use like we already had this kind of test implementation for the SAFe methodologies.	
MN	Umm. OK.	
MM	And we did start an entire new product with this, right. So, we had some teams only focused on this and other teams focused on the old methodologies, right. But then we did discovered that SAFe was working really well. So, I think that all teams in this organisation are now using this methodology.	
MN	Mm-hmm. OK, alright.	
MM	Knowing how much work actually my managers put into this and how much time, I don't think like I think it works really well for us. So, I am not sure if they would be looking for a change at this stage.	
MN	OK, alright. OK, so, my request was not specifically like a you would change, but just looking at the previous project to say OK if we had used this, how would it have worked? So, this is a form of validation for this project that I am doing.	
MM	Yeah. So, if we were not to use the SAFe methodology, you mean, and we were coming back from the previous ones.	
MN	Hmm.	
MM	Yeah, definitely. This would be this would be a good methodology to follow.	
MN	OK. So, would there be an opportunity to say maybe I would ask you to say maybe I can come there, maybe just spend a day to sit down with someone like yourself maybe and then we go through the old project that you did. And then we would see how maybe this framework would have helped in terms of defects.	
MM	Umm. I am not sure I understand the question.	
MN	OK. So, my question is that I am trying to validate this framework in a company.	
MM	OK.	
MN	OK. But the appropriate validation would be to take this and use it for a new project again.	
MM	Ok.	
MN	But you already do a similar work. But now, before you moved to Agile, you were in the traditional approach, and you had a project developed and completed for example.	

MM	Yeah.	
MN	So, in my validation I just want to say would this framework be able to work in a project that did not use agile in terms of addressing defects. So, I would be looking at an opportunity within an organization where a project was completed. Looking at this now, like in a desk-based approach to say if we had used this framework, this would be maybe the outcome. So, it's just a form of validation that I need for this framework. Much that you already have a similar thing but looking at the old project that was already completed using maybe a waterfall.	
MM	Yeah. I think if we were not to using a similar approach already, this would definitely be something that the company would test. And I am sure the that was already something that the person who implemented our methodology in our company like we did go through multiple things before we decided on one.	
MN	Mm-hmm. Umm	
MM	So, yeah, that would have been an opportunity.	
MN	Ok. Alright. OK. So, do you have any other comment for me?	
MM	No.	
MN	OK. So, M, I really appreciate. It's not easy to take time away from your busy and then attended to my request and participate in this research. I really appreciate and I ask that next time please be available as well. Because I learn a lot through interacting with the people like you from industry who are really doing these things there, yeah.	
MM	OK. Yeah. I think I did do a survey for you as well in the past, right? Yeah.	
MN	Yeah, yeah, yeah. So, that's why I am saying I do appreciate because you have been involved throughout.	
MM	No worries.	
MN	Yeah. I really appreciate. Yeah.	
MM	Yeah. If I can help in the future and I will be available. Yeah.	
MN	Umm, OK alright.	
MM	OK.	
MN	So, thanks so much Max.	
MM	Thank you. Thank you for this and good luck with your research.	
MN	Yeah. I thank you so much.	
MM	Thank you.	
MN	Alright. It has been nice, yeah.	

MM	Thank you very much as well. It was nice to meet you as well Misheck and good luck and have a have a great evening.	
MN	Yeah. Thanks so much. We will be in touch. Yeah.	
MM	Ok. Bye-bye.	
MN	Bye-bye.	

E.7 Interview 7

Position: Lead Software Engineer & AI developer

Experience in MDS: over 5 years

Interview date: 30 April 2026 9:00 to 10:37 am

Speaker	Transcript	Notes
MN	I would like really to thank you for accepting to participate in this research. The insights you will be sharing are valuable to my research.	
TH	You are welcome.	
MN	I am Misheck Nyirenda. I a m a researcher in the Regulated Software Research Center here at Dundalk Institute of Technology. And my research is about application of agile practices to address defects in medical device software development.	
TH	Ok.	
MN	Yeah. So, before we can start our discussion, would you allow me to record?	
TH	Yes. No problem	
MN	Thanks, T, for allowing me to record the conversation and your answers are confidential. No names will be used in the report.	
TH	You are welcome. I trust you.	
MN	Alright, so I shared the model. Yeah, so I will share it again on screen so that I go through it and what we expect from you. I hope you can see the screen, yeah?	
TH	Yeah, yeah, I can see it.	
MN	Okay, alright. So, this is the Agile V model for addressing defects. As I mentioned, we are trying to use agile practises to address defects. So, in this model, the practises on the left hand side of the model are used for preventing defects. And the ones on the right hand side of the model are used for identifying defects. So, at the software requirements gathering stage, we are proposing user story, onsite customer or proxy and use case. So, we're saying at this particular stage, developers can use either user story and on-site customer when gathering requirements, or they can use case and on-site customer or proxy to gather requirements,	

	or they can use all the practises at this stage to address defects. Similarly, at the software requirements analysis, we are proposing that the developers can use a combination of model storing and the behaviour driven development to address defects. They can also use pair programming, test-driven development onsite customer or proxy to address defects. But at this particular stage, we're also including a direction of implementation of these practices. So, they can use all of them to enhance defected prevention. They should be able to first use model storing and behaviour-driven development, and then follow this with TDD, pair programming, on-site customer or proxy. So, with this particular way of using them, we are trying to enhance defect prevention at this stage.	
TH	Sorry, can I just ask you quickly, what is model storming exactly?	
MN	Okay, so model storing is a practise whereby within a team, they can come up with the diagrams, for example, to explain an issue that maybe one is facing within the development team. So, for example, it could be a sketch, it could be any form of diagram that shows something that helps someone to understand exactly what an issue is.	
TH	Right, right, right. OK, perfect.	
MN	Yeah, so this would be even to the user. If you wanted to demonstrate something, you could come up with a diagram just to clarify the issues with the user.	
TH	Yeah	
MN	Yeah, so that is the implementation of practises on the left hand side. A similar approach is also taken on the right hand side, whereby we are proposing at the Software unit implementation stage, we're saying that at this stage you can also do defect prevention by using a combination of these practices: programming, TDD, model storming, on-site customer proxy, coding standards and code ownership. The application of these practices will prevent defects. So, we are proposing that at this stage they should start with the implementation of these and then follow up with the implementation of pair programming, unit testing, small release on-site customer proxy, and the refactoring. So, once they have written some code, they can implement these practises to ensure that they identify any defects within the code. Similarly, once code has been written, it must be reviewed. So, we are saying there should be code review. So, special or specific people that have got the expertise within the area in which the code applies should be able to review that code. And that would help to identify defects. And then, once the code has passed the review, as well as it has been. all defects have been identified. The next stage,	

	because we know that some defects can still link in the code, so the next stage is to implement continuous integration. So, this is a practise whereby code is continuously added to the code base. New code is added and then that process of adding that code also initiates the testing first. So, if you already had an existing code, you are adding new code to it, you need to carry out tests and you are doing rebuilding as well. So, with that process, we're saying that that would also ensure that defects are identified within that. So similarly, once this stage is completed, the next stage is unit testing. At this stage, we're saying that they should use unit testing, on-site customer proxy, pair programming and refactoring. At unit testing, what happens is that different pieces of code or modules are put together to form a system. So, in that process of putting one, two modules together, we want to test whether they are interworking properly or not. And, if not, it means defects have been identified, which needs to be corrected. So, the testers would be able to identify those defects. And therefore, if they are minor, they can easily be corrected by programmers who are supposed to be present at this stage. And also, if they are minor, then the refactoring of the code can also help to correct those defects. And then once this is done, the next stage is the integration and the testing again. So, the entire software is now integrated, all the components are integrated to form the complete system. And then, testing is also done here. So, the practises that we propose at this stage are integration testing, pair programming, refactoring, and the unit testing. Similar to what has been done here to say if any defect is identified here, if it is minor, it can be corrected. At this point, but if it is larger, it has to be referred to a specific stage. For example, if the defect is larger, it can be referred to any of the stage on the left hand side, so that the next stage, for example, if it is identified here, then it can be referred to any stage, any relevant stage in the left hand side. So, this prevents it from being an issue in the next stage in the right hand side. So, there is that referring back of major defects, even if they are identified late in the development. OK, I'm not sure if you have a clear understanding about how it works.	
TH	Yeah, no, definitely. It's becoming more clearer.	
MN	Ok	
TH	Yeah, sorry, so UAT there, what exactly is UAT?	
MN	Okay, user acceptance testing.	
TH	Okay, yeah, yeah.	
MN	Yeah.	
TH	So, this is all in line with or like this is informed by ISO 13485.	

MN	Yes	
TH	Okay, yeah, yeah, yeah. And then throughout this process, is it like you document each stage as well?	
MN	Yes, so yeah, so when developing software, documentation is very important. So, at each stage, documentation should be there.	
TH	OK. Right, yeah, yeah, yeah.	
MN	Hmm. So, what I want is that you examine or analyse each of these stages to say whether the inclusion of these practises makes sense in terms of addressing defects and whether any changes should be made on each of these stages. And also look at the complementary application of these practices, whether it would work or not, all those things. So, focusing specifically on using them to address defects.	
TH	Yeah, so should I talk about it in terms of our project development process with?	
MN	Yeah, I think that would be the next stage.	
TH	OK	
MN	Yeah, that would be the validation part to say, if we had applied this, it would have addressed defects or not.	
TH	Yeah.	
MN	So, that too, I think we do it later on. But for now, I just want you to evaluate this model to say, this model at this stage, for example, these practises make sense in terms of addressing defects or at this stage, these practises I think they don't. And they also maybe one other practise such as this would be included, all those things.	
TH	Right, yeah, yeah.	
MN	So, we start with the requirements gathering.	
TH	Yeah.	
MN	OK, maybe if you have something to clear before I can start asking the questions.	
TH	No, I mean I have no criticism of that. You know, all of this, I look at it and I was like, God, I wish we did that when we first started developing. But definitely going for the user story and having the, you know, onsite customer or, you know, proxy for that in mind, I think is very good practice. So yeah, I'm quite a fan of like really having the user be integral to your entire development process instead of just developing it and then trying to get a user to fit into what you've developed. You know, I like that user-centric focus.	
MN	Mmh. OK. So, because I have a set of questions, so I would want you to comment at each stage and also if there are specific points that you would want to make for each and every practice, that would be helpful so that I can easily write up on them.	

TH	Yeah, perfect.	
MN	Yeah, so we start with the requirements gathering. I see that you have already pointed out the importance of on-site customer, but you can also talk about the other practises as well.	
TH	Yeah. Yes, so the order practises within requirements gathering?	
MN	Yeah, these ones, like you have talked about on-site customer, but you can also talk about the user story as well as the use case.	
TH	Yeah, and when we say user story, do we mean like, you know, because I kind of was conflating on-site customer with user story like the user story what we imagine the typical user will use our medical software for, or is it something else?	
MN	Yes, yes, yes. So, they state to say, ‘as a user, I want this, this, this so that I can do this’. So, yeah. So, we're saying with that kind of thinking, is easier to catch any defects.	
TH	Yeah, I wish I could have more tougher criticism, but I just I do totally agree with that. Having the user story there is, yeah, I think vitally important. And yeah, just well, yeah, we'll talk about this later on, but yeah, in relation to our project, I don't think we had the user story totally in mind when we first started developing. But yeah, I think that's very important use case as well. Again, if these things are as clearly defined as they can be from the onset of development. I just think that's vitally important.	
MN	Mm.	
TH	Requirements gathering as well, would you have in there that like more technical, like would you also see requirements gathering as being like, I'm going to use MySQL as a database, I'm going to use Next.js as the front end. Would you define that as requirements gathering at the start or is it just kind of user focused? Yeah, okay, okay.	
MN	Yeah, so this is more on the technical side now. After you have corrected the requirements, you think of, say, okay, how can we properly support this solution for this customer? So, what technical aspects do we need? Yeah.	
TH	Yeah. Right, yeah, this is like the next step, yeah. Yeah, yeah, I see, I see.	
MN	All right. Okay, so do you think any other agile practise would be necessary at this stage?	
TH	No, for this stage, no. I think having that as your start point, I think, is the most important and focusing on that, I think, is the right thing. Nothing else comes to mind for me.	

MN	OK. Alright, then let's move to the software requirements analysis stage. What would be your opinion on the agile practises included at this stage in terms of preventing defects?	
TH	Can you just, so a behaviour driven development, what does that mean exactly?	
MN	Yeah. Okay, so you are modelling based on the functionalities or behaviours of the system that you anticipate within the project. You are more focusing on what the system is going to do, and you produce models and the looking at how those models would work within that particular system. So, behaviour in terms that we are more focusing on the functionality, how the different functions will work in the system.	
TH	Yeah.	
MN	So, that is what drives the process. That's why it is called the behaviour-driven Development. So, you are focusing on the, yeah, the functionalities. How the system is, how different functions will work?	
TH	Yeah, so yeah, you start the model storming behaviour driven development and then your like verifying that a bit with paired programming and a TDD is testable or what was TDD again?	
MN	Yeah, test driven Development.	
TH	Test-driven Development, yeah, and then, and then you still have your on-site customer in mind, yeah, yeah. Yeah, I like that approach.	
MN	Yeah.	
TH	I wonder if it like have you thought about, you know, obviously we're like living through the age of large language models now. To you, is it acceptable at all to do this, you know, model storming and driven Development and thinking about the requirements you need with a large language model. You know, is that acceptable to you? Is that something that should be considered now? Because, you know, I know they can hallucinate, but, you know, by and large, they do have a very good idea of what's out there and they have search capabilities and they can really test. And should that be a part of the requirement analysis at all?	
MN	Yes, so at this point, we are not focusing on the AI part.	
TH	OK.	
MN	Yeah, I feel like that would be a different area on its own. So maybe, for example, in the future, people would say, OKane, how could we integrate AI using this particular model?	

TH	Yeah, or not even integrating, I don't mean integrating AI, but using AI to...	
MN	To put down the requirements?	
TH	Yeah, to ask just like, hey, I'm thinking about building a front end that does this. What would you recommend as, you know, to use, you know, and then and then it tells you, okay, Next.js is good at this, you know, React is whatever, you know, it is. Yeah, I don't know, that's all I can think of there, because I mean, that is, I'm a kind of one man development team, so it was very useful for me to actually have a conversation with AI about the software requirements, you know.	
MN	Ok.	
TH	But I do really like this approach in general. I like you know, the idea of model storming, behaviour-driven development, and I like that there's like verification kind of processes there. Yeah, and again, just that you keep mentioning the on-site customer, I think is very important.	
MN	OK, yeah, alright. Apart from the suggestion to use AI. Any other agile practises that you feel would be fit at this at this stage to address defects or to prevent defects?	
TH	No.	
MN	OK. Alright. Then let's move to the software architectural design stage. What would be your comment on the agile practises that are included at this stage? In terms of preventing defects.	
TH	Yeah. Model Storming again. Integration testing.	
MN	So, integration testing, maybe I should clarify here because I think the traditional way of thinking about it is to use it later on. Like the way it appears here in the software integration and integration testing. So here what we are saying is, because this process is done by software integrators or integration testers. So, by including the practise here, we're saying that integration testers should be part of this process of software architectural design. So, as the architectural designs are being developed, the integration testers would see whether or how these architectural designs will integrate, will integrate and work together. And with that inclusion of these people, defects would be easily identified or found. And then they would be fixed earlier than waiting until late when they will be involved late on later on.	
TH	Yeah. No, Ithink that's very good practice in catching defects. Yeah, and then you mentioned a simple design there as well. So yeah, when you say simple design, I know that's	

	an obvious statement, but you mean like yeah, I don't know, how would you define that?	
MN	Okay, so, we're saying the design should not be complex to allow the developers later on to easily understand what they're supposed to develop, because you give this design to the developers later on. So, if it is complex, it will be difficult for them to figure out what they are supposed to development and accomplish, but the agile specified that, as much as possible, the design should be simple so that everyone in the team can easily understand it.	
TH	Yeah, yeah.	
MN	So, the thinking is, if it is complex, there is a high chance that it would lead to defects. But if it is simpler, it means people would easily understand because the ambiguity is a very big issue.	
TH	Yeah.	
MN	Defects of ambiguity, where people don't understand and they develop things that are not supposed to be what the customer wants.	
TH	Yeah. No, I really like thinking, or like the thinking about this like it's all lowering the probability of defects. But yeah, I think, I don't know if I have much more to comment there. Like I really like the idea of that you, well, yeah, with the simple design maybe come before, like the first integration testing. Because I like the idea that yeah, you model storm, you have the onsite customer in mind, you come up with this initial simple design and then you like, you know, integrate, do it integration testing just to see in this simple design, do all these parts fit and.	
MN	Mmh	
TH	Yeah, if they do, you have this like very good foundation to build upon.	
MN	Okay, so you are proposing that the simple design should be done first here?	
TH	Yeah, maybe, you know, in my in my naive sense, I mean maybe that that would be something.	
MN	OK. Yeah, okay, alright.	
TH	But yeah, but I do really like the process behind all of this.	
MN	Okay, so maybe just one question. Do you think that the practises included at this stage would help to prevent defects?	
TH	Yeah, definitely, definitely they would.	
MN	OK. Any more comments at this stage?	
TH	I don't think so.	

MN	Alright. Then let's move to software detailed design stage. So, what would be your opinion on the agile practises included at this stage in terms of preventing defects?	
TH	So yeah, again, I kind of like the idea that you, let's say you're developing the front end, that you also have like, come up immediately with a simple design there, because I know we didn't do that from our own front end and it got very messy, so I like that.	
MN	Mmh	
TH	You know this idea of like come up with something simple first, you know, along with the model storming process and keeping your on-site customer there. And then, yeah, what, so in terms of if this is more detailed design, what would integration testing be here? Would you have like a practical example of that?	
MN	Yes, so at this stage, detailed design, we're looking at individual modules. So, the inclusion of integration testing here is similar to what I mentioned to say we are talking about integration testers. So, if you have individual modules, the integration testers will have a clear understanding of each one of these modules at this point. And therefore, they can easily think of how they are going to test the integration of these modules later on. So, we see making everything clear beforehand rather than later on to say they don't have any idea and then they just testing the integration later on. But at this point, they can easily also figure out to say, think this and this module will have issues, these issues in terms of integrating them together. And therefore, you can address that before, later on, before the code for each of those modules is written.	
TH	Yeah, yeah. No, again, similar to before. I really like this. Yeah, maybe the same comment of like the simple design for me could be more forefront. Other than that, I really like it.	
MN	OK. So, just to just to confirm, do you think the agile practises included at this stage would help developers to prevent defects?	
TH	Yeah, yeah, they would.	
MN	OK. Then let's move to the software unit implementation. Would you give your views on the practises included at this stage in terms of addressing defects?	
TH	Yeah. So, when you were talking through before, you mentioned continuous integration. Just remind me, so, like, what did you fully mean by continuous integration? You like develop code and like implement that with all the kind of. I don't know if surrounding documentation or?	

MN	Yeah, yeah. So, what happens with the continuous integration is that code is continuously added to the codebase. So, you have codebase where code that has been developed and tested and is working is. But you are continuously adding new functionalities to the already existing code, so continuous integration is the practise that allows that. So, when you have done, for example, completed a new functionality, you add it to the existing codebase, and that process requires that you build and test. So, its cycle on its own. Build, test, build, test. We make sure that the new added code does not break everything that is already existing.	
TH	Yeah, no, I mean it all seems, just from the outset here, overall, it seems very good to me, and I couldn't think of more agile practises to prescribe to that.	
MN	OK. Any practise in particular that maybe you would recommend using at this stage? From the ones that are included, are there any practises that you would say are crucially important at this stage?	
TH	Yeah, well, Code review maybe jumped out at me. I do think, yeah, that's quite important. Even just to get a like throughout, especially in the age of like AI development, you can really like lose track of your code and not be aware fully what's there and what it is and how it interacts with everything. So even if you're developing with AI, I think like code review is quite important. Just for your own awareness of the project. So yeah, I don't know, that's something maybe I would just mention or highlight.	
MN	Ok	
TH	That, yeah, you know, anyway, it's just very easy to use or not have a real sense of the code anymore, so I think maybe even code review is more important now than ever. I think it's very good you've coding standards and code ownership in there that's really important. You know, you don't just implement something without these things in mind. I think that's very good. Yeah, unit testing, small release, very important, refactoring, code review, like this cycle, I think. I don't think of any comments or criticisms. I really like that all these things are. there and implemented and yeah, you do that for each unit you implement. So yeah, I don't know. I guess that's my comment on it.	
MN	OK. Alright. Any more comments at this stage?	
TH	Don't, I don't think so.	

MN	Okay, alright. Then we move to software unit testing. What would be your comment on the use of agile practises at this stage to address defects?	
TH	So, yeah, it so says unit testing, but I suppose writing a test and implementing a test like that all has best practises as well. I don't know if it's relevant to this at all, but I just, I feel it's like you can lazily write a test and you can like give it real thought. and I'm sure there's like, you know, practises on that. I don't know if that should be referenced here or anything, but that's just something that came to mind that like, you know, I'm sure testing is a kind of discipline in itself and it's very important to be good at writing good tests. and implementing that. So, I don't know if that should be more expanded on or something like that.	
MN	OK. Okay, any more comments.	
TH	No, I don't think so.	
MN	Okay. What would you say about using refactoring at this stage?	
TH	You get a more, you know. Hopefully, true refactoring, you get a more kind of, let's say, condensed code; it's easier to understand, it's, you know, works better, and so all those things are great. Code review should be there as well or something like I feel like that's a consequence of refactoring that it's almost a code review in a sense, I hope that makes sense.	
MN	Okay, so you are recommending also to use code review at that point, yeah?	
TH	Yeah, yeah, let's say that.	
MN	OK. Then we can move to the software integration and integration testing stage, yeah?	
TH	Yeah.	
MN	Okay, so what are your views on the other practises included at this stage to address defects or to identify defects?	
TH	Yeah, so Integration testing, PP, refactoring, unit testing. I think all that is really good. Continuous integration as well. Yeah, and you have the on-site customer and your team. Yeah, I actually look, I'm pretty, I'm pretty, I don't think of any comments there. I think that's quite good.	
MN	And so, you think the agile practises at this stage could be identified defects?	
TH	Yeah, the would.	
MN	OK. And in terms of their complementary application, what would be your opinion?	
TH	The sorry, the complementary	
MN	Yeah, like the combined use of these practises at this stage.	
TH	OK. All those practises being used together.	

MN	Yeah, yeah, yeah.	
TH	Ilike, I think, yeah. I think those specifically would be very good at identifying defects. You know, it's like multi-kind of factorial approach. The more I think of refactoring, you reduce the risk of defects, to unit testing, you reduce the risk of defects, and I think altogether you have a combined effect of reducing defects, so I think it's quite good.	
MN	Mhm. Okay. Any other practise that you think would be useful at this stage identify defects?	
TH	No, I don't think so. I don't think so.	
MN	Okay, let's then move to software system testing stage. What would be your views on the agile practises used at this stage to identify defects?	
TH	I'm sorry with this. Can you just clarify UAT for me again?	
MN	Okay, so it's user acceptance testing. So, this is the testing done on the user side to see, to verify if what they specified earlier on as their requirements have been met.	
TH	Ok. Yeah, I really like that. That's a part of both of those stages. Yeah, no, I don't think I have any comments on that. I think the UAT is great to have there. So yeah.	
MN	OK. And in terms of the complementary use?	
TH	Yeah, again, it's great to. Yeah, combining UAT with the paired development and refactoring and the unit testing is just, it's a strong approach. Yeah, if you can really implement that. So, I think that's good.	
MN	OK. Any other practise that you think would be appropriate to use for identifying defects at this stage?	
TH	No, I don't think so.	
MN	OK. Then we move to software acceptance, acceptance testing stage. What would be your views on using agile practises at this stage to identify defects?	
TH	Yeah, I like that you view it first there. Yeah, I think for acceptance testing, it just has to be UAT led, and then I like that you have the like secondary check almost of PP there, and yeah, you still have on-site customer or proxy there. So, I think it's quite good as is. I don't think I could think of more to add there.	
MN	Ok. So, what would be your overall comment regarding the AV4AD model in terms of addressing defects?	
TH	Yeah, I think it's really strong for addressing defects and I do really like that there's, I think at every stage, multiple ways of catching defects, which I think is one of the strongest elements of it and it's clear.	

	And. Yeah, I think it's a strong model for not only catching defects but preventing defects in the 1st place. I think that's, yeah, a core strength of it.	
MN	Ok.	
TH	It's a really nice kind of like workflow if you want to call it that.	
MN	OK. And what would you say about using this, the AV4AD model in relation to the IEC62304 standard?	
TH	Yeah, my knowledge of the IEC62304 isn't too strong, but I am sure, yeah, I think it's good to use that, yeah.	
MN	So, the IEC 62304 is about the software development lifecycle processes. So, what we have done with this model is to take all these stages that you see have been adopted from that standard. So, it is the software development life cycle processes as the requirements gathering, software requirements analysis, architectural design, detailed design, up to acceptance testing. So, what I've done is to adopt those into this model. So, do you think it this model aligns with this standard, and therefore, if the developers use this model, would they be in compliant with this Standard?	
TH	Yeah, it seems to me that they would. Yeah, it seems to be that at each stage, aligns with something there and Yeah, it's very useful to do this sort of practise so you're immediately in line and you don't have to be kind of retrospectively in line. You don't have to come back after the fact and try to be in line. I think this, yeah, this does align with it.	
MN	OK. So, in terms of the development approach in your project. What development approach did you follow?	
TH	There was no specific development approach. It was just see if it doesn't break. If it doesn't break, great. And keep moving forward.	
MN	Ok.	
TH	So yeah, I wish I could, yeah, I don't know tell you, but like, you know. It was kind of I mean, there, you know, maybe vague resemblances to some of these stages, like, so for requirements gathering, it was like a long discussion with a large language model of, you know, what would be best. I'm planning this, you know, and then, you know, we settled on kind of Django and Redis and Celery for handling all these kind of user queries. So, there was this kind of requirements stage, but what it didn't have was a user-centric approach. You know, we weren't, we just had no idea that it would be useful and clinicians might like it, but we didn't have clinicians there to actually kind of, you know, talk to and feed us back in for information and for them to test stuff. So,	

	it lacked that, but then, you know Yeah, it did have the requirements gathering stage in that way. It did have like model storming in the sense of, you know, again, kind of with a with a large language model and there were there were a few people working there and then.	
MN	Mm-hmm. Ok	
TH	Yeah, I don't know. I did have, yeah, I don't know. And I think what it's lacking is testing a lot and writing good tests. But anyway, sorry, I'll stop. I'll stop rambling about it now. I'll be more focused.	
MN	OK. So, what changes would you suggest in the AV4AD model? So that we improve it.	
TH	So, yeah, I mean, the just to like, so this model, let's say the target audience of this model is like established software engineers who are developing medical software.	
MN	Yeah, so firstly, we're focusing on anyone who is thinking of developing a medical device software to say they should be following this, by following this model, they should be able to reduce the likelihood of defects in their software.	
TH	Yeah, well, maybe what I could add then is just kind of like, you know, my own kind of perspective, which is you do have a lot of PhD students who don't have good software engineering background, and they are trying to develop, you know, medical software. So, the only thing I think this is a really great model.	
MN	Mmh	
TH	The only thing is, you know, it's, I think, only intelligible to a established software engineer. So, or like someone who's really familiar with, you know, the lingo and whatever. So, from my side, maybe the improvement could be that it.	
MN	Mmh	
TH	Just could be more understandable to someone from a non-software engineering background because a lot of people from a non-software engineering background are developing medical software and even more so with AI now. So maybe that's the only improvement I could say. But as a model, I think it's a very good model.	
MN	OK. So, now that we have discussed the model and we have provided some of the suggestions, what would be your thoughts on adopting this model in your development process?	
TH	Yeah, I would. I would definitely adopt it. I think. We really lacked a lot of these processes, and I wish we had, you know, done them at the time, so going forward, I would. I would adopt this workflow for sure.	

MN	The other question would be, would you recommend using this model in your organisation to address defects?	
TH	Yeah. Yeah, we're not only developing the thing I'm developing, but we kind of do a lot of code and a lot of development. So, I would recommend it to my coworkers as well, who again aren't established software engineers. So yeah, I would recommend it to them.	
MN	Alright, and so again, what would be your overall opinion on this model?	
TH	Yeah, again, that it's good to have a clear representation of something like this, and to visually see it is great, and I think it. Yeah, it clearly. You know. Yeah, I think it clearly demonstrates how you can prevent defects in in the development process of medical software and has the joint effect of being in line with established standards. So, it's, yeah.	
MN	Ok. And do you have anything else that maybe you would want to add?	
TH	No, I don't think so.	
MN	OK, alright. So, I think I have concluded the questions that I prepared for the interview. Now, the other part, as I mentioned, I am doing 2 parts, the interview and the validation. So, the interview is the evaluation, which we have done, and you have provided feedback, which is very useful for me. And then the other part is the validation, whereby now we want to go through your development process that you followed to say if we followed this one. So, we will be applying this model to say, OK, now we are at the requirements gathering stage and therefore we need, I would recommend maybe using this bad practise and then you'd be stating whether that practise would have helped to address some defects that maybe you faced in the development process, in your initial development process, to say, okay, if I had used this per practice, it would have addressed this particular defect. Yeah, so that's what I want to do. So we go through the whole process, the whole model at each stage. So, we can now we're moving to this stage and at this stage, I think in my development process, I would have done this, followed this particular practice, it would have done, collected or hoped to prevent this particular defect which we faced all those.	
TH	Yep. Perfect.	

E.8 Interview 8

Position: Software & Mechanical Engineer

Experience in MDS: over 5 years

Interview date: 1 May 2026 2:00 pm to 2:33 pm

Speaker	Transcript	Notes
MN	I would like to thank you for accepting to take part in my research. The insights you will be sharing are valuable to my research.	
BC	You are welcome.	
MN	I am Misheck Nyirenda. I a m a researcher in the Regulated Software Research Center here at Dundalk Institute of Technology. And my research is about application of agile practices to address defects in medical device software development.	
BC	Ok.	
MN	Before we start our discussion, I would like to request your permission to record?	
BC	Yes, you can record.	
MN	Thanks, B, for allowing me to record and your answers are confidential. No names will be used in the report.	
BC	That's fine.	
MN	Alright. We can start the interview now. So, we will start by first, maybe you providing your background and experience in software development.	
BC	Yep. So, I am actually qualified in mechanical engineering and software engineering. I have done several software projects and really enjoyed them. I got involved in a medtech startup. I also used some open source computer vision models to develop computer vision technology. I have also done a medical robot. I have built Benchmark Sports, which is a software platform, so mobile development and platform development, web platform development, and brought that to market. And now I am working for another medtech startup as well. So, it's platform development as well.	
MN	Ok. So, I will share the model that I have developed so that I can provide a bit of explanation and then we can start the questions.	
BC	Okay.	
NJ	Great. And Misheck, just for the purposes of the recording, we're both working as well on device X. So that's a cognitive deficit medical device.	
MN	Oh, okay.	
BC	Yep. Some do not know as of recently.	
MN	Okay, yeah, so just be flexible with me. So, whenever I need more information, I can just reach out to you, yeah?	
BC	Of course, yeah.	

MN	So, this is the model. I hope you can see the screen, yeah?	
BC	Not yet.	
MN	Okay, let me share. Yeah, can you see it now?	
BC	Yep.	
MN	Okay, alright. Yes, so, were able to seem to see me on the camera, was my camera on or not?	
BC	Yes, it's on, yeah.	
MN	OK, alright. Yeah, so this is the Agile V model for addressing defects and the development of this model incorporated stages which are adopted from the IEC 62304 standard.	
BC	Mhm. Yeah.	
MN	So, the stages, requirements, software analysis, up to acceptance testing are adopted from that standard. In that standard, they are called life cycle processes, software life cycle processes. And the	
BC	Yeah.	
MN	So, they have been incorporated in the V model. As you can see, the structure is the tradition of V model. And therefore, the inclusion of agile practises within this model, that's where the name comes in to say agile V model for addressing defects. So, the agile practises included on the left side of the model are for preventing defects. Those on the left side are for identifying defects. So, identification happens after much development has been done and therefore, we are trying to find the defects later on.	
BC	Yeah.	
MN	So, that's where that's what identification identified defects means the while prevention. We are trying to, as much as possible, to prevent the defects or to ensure that defects are not introduced in the software as early as possible. So, this happens in the early stages, starting with the software requirements gathering, software requirements analysis, up to software unit implementation. So, what I want you to do is to review this and provide feedback. We are going to look at each and every stage. So, you give feedback on each one of the stages based on the agile practises included there, specifically focusing on their use for addressing defects.	
BC	Yeah. Yeah.	
MN	And if there is anything that is maybe overlooked at a particular stage, then you need to point that out. And also, maybe a guide on how we can improve at that particular stage.	
BC	Mhm. Okay.	
MN	Yeah, so, are you okay that we start?	

BC	Yep, yep, and I'm just thinking the first two requirements, gathering requirements, analysis	
MN	Okay, alright.	
BC	Yeah, so is this like a finalised model or is this being developed in or in terms of okay, I just Like I come from trying to make many startups, some not so good, some good. And I've went through the first stages many, many times.	
MN	Mmh.	
BC	My immediate reaction is that there's something missing at the early stages?	
MN	Mhm.	
BC	Like requirements, is it is it meant to be like a broad title? Is it? Is it is a title meant to be broad of each of each section?	
MN	Uh, like, which, which one?	
BC	So, requirements gathering and requirements analysis, and software requirements analysis, and then requirements gathering. I just think it's very broad, maybe to then jump into architectural design. Like I think the first two points could be extended a lot more. And I also think that there's something missing before gathering. I would have thought analysis would be nearly before gathering because it's kind of like a long process of all the potential things. I know it's not really; it's not meant to be creating a solution here. This process, this system you've designed, but I think it's just the initial stats are something missing.	
MN	Mmh okay.	
NJ	And B, do you think it would be planning? Because in the IEC62304, it's planning that comes before, really.	
BC	Yeah. Yeah, planning, and even before planning is kind of like identification of something that's missing or like you have to like before you even get, if you don't know what to gather, you don't know what to plan. There has to be like a point where it's like, you've had something like, well, there actually is something here.	
NJ	Feasibility. Yeah, it's a good point. It's a good point, but it might highlight the fact that, and actually this is very important and possibly critical, but 62304 is founded on the other standards. So, do you remember when we talked through the user needs? And I said, they're out of scope of 62304. But because they're out of scope, it's the product level user needs and, you know, identification of technologies and feasibility and all of that kind of stuff that's done outside of IEC62304.	
BC	Yeah. Okay	
MN	Yeah.	

NJ	Which actually, to your point, I think is highlighting. That, you know, you're right, that there is something missing, I suppose, yeah.	
BC	Yeah. Yeah, yeah, that's exactly the thing I was thinking about, like that whole process of like actually building something that's actually needed. It's just not really mentioned there in the first few steps.	
MN	Hmm. Okay. Mhm.	
BC	So, obviously, outside the 62304, whatever, but it, yeah, probably shouldn't be. I don't know, like I'm no person to talk about standards, but that's just my immediate reaction.	
MN	Okay.	
NS	Well, I just updated 62304, the CD2. That's the draft that's going, that's intended to go out to public for the second comment. So, I suppose I'm the right person to comment on from a standards perspective.	
BC	Yeah.	
NJ	I think from that model, the feedback that I think both of us agree on, B and myself, is that you could at least put a disclaimer to say that defects could potentially arise from the higher level standards that are intended to feed into 62304. So, I think that's a good catch.	
MN	Mhm. Mm.	
BC	Mm. Yeah.	
MN	Okay	
NJ	Well done, B.	
BC	No problem, happy to have something.	
MN	Mhm. Yeah, right. Okay, thanks so much for that. So, looking at these stages that have been put in this model and focusing specifically on the agile practises that have been included, could you please comment or provide your overview in terms of these practises for addressing defects.	
BC	Yeah, so the beyond, so like architectural design, detailed design, unit implementation from that point onwards, is that the point you're talking about?	
MN	No. So, at the requirements gathering stage, practises included there are user story, on-site customer, and use case, yeah.	
BC	Okay. Okay, yes, that's sort of what I was talking about there, but requirements on user story, onsite customer, yeah, and yeah, just really iterative process, I can guess, yeah, the circle's there, that's gonna have to be iterating a lot before you move on. Like, I thought circle was bigger than the rest, and yeah, yes, I agree with that, yeah.	
MN	Okay, and in terms of their complementary use for addressing defects at this stage, complementary in terms of using them all together.	

BC	And, sorry.	
MN	Like these practices, user story, onsite customer and use case, using them together at this stage to address defects. What would be your comment?	
BC	Yeah, I suppose they should probably be separate. Yeah, definitely. And the user story would definitely be a different step to the use case because the story is only understanding the story and then finding what the use case actually is another part of the journey. And, yeah, I would probably see that in separate parts, and is there your requirements gathering to?	
MN	Mmh. OK.	
BC	Next step, like if all the requirements are so many requirements, but you only pick one thing and to move on, so maybe it would be a separate step, maybe, yeah.	
MN	Mhm. Okay, any other agile practise that you feel would be appropriate to use at this particular stage?	
NJ	I can't think of anymore, M offhand.	
MN	Okay. All right.	
NS	It'd be handy to have the full list of agile practises to know have you excluded any? Do you know that why have you selected some and maybe not others or have you got all? Do you know it would be helpful to know that?	
MN	Mmh.	
NJ	Do you have a comprehensive list of the different agile practices?	
MN	That could apply at this stage?	
NJ	Yes.	
MN	Yeah, so these are the ones that through the analysis were recommended for this stage. So, I'm just asking based on the other experience to say if they know anything more in terms of agile practises that, maybe I overlooked.	
NJ	Ah. Gotcha. Yeah, thanks. I don't.	
MN	Okay. B, do you have any agile practise that you would recommend using at the software requirements gathering, specifically for addressing defects?	
BC	No, I think it's all good.	
MN	Okay, alright. So, we move to the software requirements analysis stage. What would be your opinion on the agile practises included at this stage for addressing defects?	
BC	Mm.	
MN	So, we have model storming, behaviour-driven development, pair programming, TDD, as well as onsite customer or proxy.	

BC	Yeah. Yeah. What do you call it, sorry? I'm not familiar with this terminology. What is it, sorry? You give some terminology for the agile. What was it, sorry?	
MN	I said, Agile practices.	
BC	Agile practises?	
MN	Yeah. Do you think this practise would help to prevent defects at software requirements analysis stage?	
BC	Mm. Yes, I think that's all good, and there's nothing I can add to that point, no, that's perfect, yep.	
MN	Okay. Also, at this stage, there is the direction of implementation to say developers should start by using a combination of model storming and the behaviour-driven development, and then this should be followed by the use of Pair programming, TDD, and on-site customer proxy. So, what would be your comment on this direction of implementation?	
BC	Pair programming, TDD, and on-site customer proxy. So what would be your comment on this direction of implementation?	
MN	Okay, so do you think this kind of implementation could help to prevent defects.	
BC	Yeah, I think that's good, yeah.	
MN	Okay. Do you have any other comments on this stage regarding agile practises use for addressing defects?	
BC	No, all good, that stage.	
MN	Okay. Alright. And then we can move to the software architectural design, yeah.	
BC	Yep.	
MN	Okay. What would be your views on using these agile practises at this stage? So, we have model storming, on-site customer or proxy, integration testing, and simple design	
BC	Eh. I'm not sure architecture design. I don't understand the integration test in part being not part of the flow and if it hasn't been created yet? Unless you are designing the system there, yeah, unclear the fact saying design the architecture, but you're actually designing like you're creating the architecture and maybe a design. Architectural planning, architectural design, and with a better way of to split that up.	
MN	Okay. So, maybe just to clarify, the use of the integration testing at this stage is to mean the inclusion of integration testers to say when an architectural design is being developed, integration testers should be there and the that would help them to understand clearly how different components would fit together. And they can easily identify some of the things that would not work.	

BC	Yeah, yeah, yeah, that would be great. Yes, yeah.	
MN	Yeah.	
BC	I agree with that.	
MN	So, any comments on using these practises in preventing defects?	
BC	Yeah, no, it makes sense. Just it, it, on my first glance, kind of made me believe that you're designing the system, but you're also not there. What you're saying is that you're designing the system while you're planning on designing the system at the same time as what you're essentially doing there with the, yeah, that makes sense, yeah.	
MN	And what would be your comment on using model storming at this stage?	
BC	Yeah, I think it's a wee bit of a separate step, but maybe, and yeah, when you explain it to good, yeah, it makes sense, but I imagined it was a planning first and then design second, but yeah, I suppose you can include it in one step, yeah.	
MN	Mhm. And what about the use of simple design for addressing defects at this stage?	
BC	I'm not sure what that means, really, sorry.	
MN	Yeah, so, it’s just making sure that the design is simple so that the stakeholders can easily understand what is supposed to be achieved.	
BC	Yeah, yeah, definitely, definitely simple design at the start, yeah.	
MN	Okay, and on-site customer or proxy using it at this stage for addressing defects?	
BC	And yeah, that needs to be a requirement, definitely.	
MN	Okay. Any other practise that you would want to include at this stage for addressing defects?	
BC	Nope, all good for me. Yep.	
MN	Okay. Then we can move to the software data design. So, what would be your opinion on the practises included at this stage? So, we have model storming, on-site customer proxy, integration testing. There is also a direction of implementation to say first you start by model storming, onsite customer integration testing, and then you follow that with the simple design on-site customer and the integration testing.	
BC	Yeah, that's good. Obviously, I'd circle back verification steps, massive in terms of the next three steps and design information, unit testing, design information, yeah, and refactoring. Yeah, they're all related. It's nearly all one step. And if detailed design was the overall process, nearly got a big circle and the parts were within it nearly, rather than being an individual circle looking like it's separate.	

MN	Mmh.	
BC	And. Or detailed design and implementation and have multiple steps with them - that's like a big circle, yeah.	
MN	Okay, any practise that you feel would be also suitable for addressing defects that software-additive design?	
BC	And I've not had that one, no.	
MN	Okay. Any other comment that you have regarding addressing defects at software data design?	
BC	Nope, all good. Nothing out there.	
MN	Oh, got it. Then we move to software unit implementation. What would be your comment on the software unit implementation regarding the use of agile practises for both preventing as well as addressing identifying defects?	
BC	No, again, I think the points, I wouldn't have anything to add on the stage specifically, so we can skip through the three of them, but what I would say there is that I I would include the three pieces as one piece as how I see things, and so there's no I have not had to add on software detailed design, software unit implementation or software unit testing. I think that's one thing with three parts. That's how I would do that.	
MN	Mmh. OK, and in terms of the practises that are used there for addressing defects, what would be your comment?	
BC	Yeah, no, small release, absolutely. That's way small changes, testing, small changes, testing, small changes, testing. Refactoring is obviously something that might be needed, but you'd hope it's planned well enough. That's not too big of a thing. Yeah, I think that's a big circle.	
MN	Mhm. Okay, what would be your comments on me using code ownership in terms of addressing defects.?	
BC	Code ownership. I'm not too sure of the handle out there, not, I'm not really out to that point.	
NJ	I wouldn't either. The only thing I was wondering about, I'd agree that most people these days use Agile and they are doing the unit and integration and if they do unit at all, because they don't have to for low risk devices.	
MN	Mmh.	
NJ	So, they do the integration and system testing all together, and they usually call it CICD, continuous integration, continuous development. So, I do definitely believe that the V model is outdated and that feedback has gone back to the standards committee. Which is why under CD2, the next version of the standards, they are, well, I wrote actually the agile section, and it reflects the TIR, the AMMI TIR 45, I think the number is now, can't remember offhand, losing the memory.	
MN	Okay.	

NJ	I definitely think that's an opportunity to remove defects. But yeah, like B, I don't have any idea about the code ownership.	
MN	Mmh.	
NJ	You know, it's a cross-functional team. I do, I do think pair programming, and you know, as B said, refactoring with someone else independently is very helpful, but that's all.	
MN	Yeah. All right, so you have at least I think he commented on the software unit implementation as well as the software unit testing.	
BC	Yep	
MN	So, we can move to software integration and the integration testing.	
BC	Yeah.	
MN	What would be your views on using the agile practises at this stage in terms of identifying defects and preventing defects?	
BC	I might have to answer the three of these in one go, just with the interest of time. M, I got a few things I have to do and might head on a plan for 30 minutes. I don't want to waste your time either, but I'll take a second to think how i can answer the three of them together, because if it's gone step by step, I just, I think a bit broader than that.	
MN	Okay, okay. Yeah, yeah, that's fine, yeah.	
BC	Let me just take a second to think.	
MN	Don't have anything other than I find them all together.	
BC	Yeah, one thing I would say is just, I'm a big fan of, I don't see anything to do with, I don't know if it's included, but branching, development, staging, and then production. I don't know if that's in any of them shortened terms there, but I think that's huge there and that it would integration is using staging on a staging platform and the staging platform can be used and development is obviously done in development and then the production is obviously when it's after being through this process twice. on your development on your staging.	
MN	Mhm. Mm.	
BC	That would be huge for D. I suppose it's not defect prevention but rather preventing defects in the production and integration and testing. So, on that point there, I would say integration and integration testing should be done on a staging environment.	
MN	Mhm.	
BC	I don't see anything mentioned there, and then acceptance testing that whole thing should be done on to staging and maybe the final part of that is like production.	

MN	Okay.	
BC	That's what I would add on them.	
MN	Okay, so yeah.	
BC	So, maybe for me overall, just looking at the thing in a full picture, I think there's two sets missing. At the stop, just if I look at the full thing as a whole, I think there's probably three points for me overall just to conclude my thoughts on the whole thing. At the top, there should be something a bit more in before requirements gathering, then the bit at the bottom of the triangle. detailed design, unit information, unit testing, verification could nearly be like a circle, like a bigger circle that includes three of them. And at the end, I think there's something missing in terms of accepting testing towards production and the actual switch from staging environment to production. And there might be steps involving that as well.	
MN	Mhm. Mhm.	
BC	because obviously you'd have to make sure your whole thing is set up so they can move from stage into production. That's what I would add to that, if that makes sense.	
MN	Okay, so just to mention about the, yeah, I agree that the staging processes, that's where you need to carry out these practices, these processes, like software system testing and acceptance.	
BC	Yep.	
MN	But could you comment on using the indicated practises within those stages in terms of addressing defects?	
BC	I think the refactoring at system testing should not be in that stage as you go through the whole process again. Refactoring should be done before that. I see refactoring multiple times, cannot mention everyone really. But I think when you're at system testing stage, refactoring shouldn't really be a thing, you're testing, you know.	
MN	Okay.	
BC	Because if you refactor, I think you're earlier in the process, because that's a big potential for.	
MN	So, at which stage would you recommend specifically refactoring?	
BC	Every stage between detailed design. I think the refactoring should be in the bottom circle. So detailed design, unit implementation, unit testing. I don't think it should really be anything involved that. I see it's in integration testing and system testing. I don't think refactoring should be there.	
MN	Okay, alright. What about the other practices at this stage?	
BC	What's UAT?	

MN	User Acceptance Testing	
BC	User acceptance testing on PP, Pair programming	
MN	Mmmh.	
BC	Yeah, yeah, I think my biggest takeaway from last stage is just for refactoring in the wrong place, in my opinion. That's my last one.	
MN	All right.	
BC	Apologies, you probably have too much to add, M, but hopefully you got something about that. I'm going to have to head on here. I just have a few things I have to do, but to conclude.	
MN	Okay, all right. No, for me, I appreciate Brian. The information you have provided is very useful for me. Yeah. And for you taking the time to attend to me, it's something that I really, really appreciate.	
BC	No problem at all. Happy to help. Happy to help. No, no problem at all, no problem at all. I have family of people in academia, my brother's PhD, so I understand the hard work you're doing, so best of luck with it all and all the best.	
MN	Alright, thanks so much.	
BC	No worries, M.	
NJ	I live, right?	
BC	Alright, N, I got sorry your voice is cut now then.	
MN	Yeah.	
NJ	Thanks. Take care.	
MN	OK, thanks so much.	
BC	All the best. Bye.	
NJ	Oh, sorry. Bye-Bye.	

Appendix F: Code Book

Name	Description	Files	References
Affirmed Model for addressing defect	Interviewee affirms the AV4AD model addresses defects	6	14
Architectural Design	Assessment of practices for preventing defects at Architectural Design stage	6	25
Affirmed Complementary	Interviewee affirmed complementary use of practices for preventing defects	3	6
Affirmed Individual	Affirmed practice for preventing defects	6	12
Undecided	Interviewee is undecided about a practice to prevent defects	4	7
AV4AD Adoption	Interviewee indicates that they would adopt the model in their organisation	6	10
Comparison	AV4AD comparison to current approach	2	5
Detailed Design	Assessment of practices for preventing defects at Detailed Design stage	6	22
Affirmed individual	Affirmed practice for preventing defects	6	14
Affirmed complementary	Interviewee affirmed complementary use of practices for preventing defects	4	4

Name	Description	Files	References
Undecided	Interviewee is undecided about a practice to prevent defects	3	4
Improvements	Suggested Improvements to the model	5	23
Inclusion of standards	Interview proposes inclusion of standards in the model	2	12
Integration & Integration Testing	Assessment of practices for identifying defects at SW Integration & Integration Testing stage	4	19
Affirmed individual	Affirmed practice for preventing defects	4	15
Affirmed complementary	Interviewee affirmed complementary use of practices for preventing defects	1	1
Undecided	Interviewee is undecided about a practice to prevent defects	2	3
Methodology	Methodology currently used	5	10
Model aligns with IEC62304	Conformance to standards	6	8
Practices in AV4AD	Feedback about additional practices	1	4
Practices to readily adopt	Interviewee indicates practices they would immediately adopt in their organisation	1	4

Name	Description	Files	References
Recommend model	Interviewee recommends model for MDS development	6	6
Requirements Analysis	Assessment of practices for eventing defects at Requirements Analysis stage	6	39
Affirmed Complementary	Interviewee affirmed complementary use of practices for preventing defects	6	19
Affirmed individual	affirmed practice for preventing defects	5	15
Undecided	Interviewee is undecided about a practice to prevent defects	3	5
Requirements Gathering	Assessment of practices for preventing defects at Requirements Gathering stage	6	47
Affirmed individual	Affirmed practice for preventing defects	6	27
Affirmed complementary	Interviewee affirmed complementary use of practices for preventing defects	5	14
Undecided	Interviewee is undecided about a practice to prevent defects	1	6
SW Acceptance Testing	Assessment of practices for identifying defects at SW Acceptance Testing stage	6	16
Affirmed individual	Affirmed practice for preventing defects	6	12

Name	Description	Files	References
Affirmed complementary	Interviewee affirmed complementary use of practices for preventing defects	3	3
Undecided	Interviewee is undecided about a practice to prevent defects	1	1
SW System Testing	Assessment of practices for identifying defects at SW System Testing stage	4	11
Affirmed individual	Affirmed practice for preventing defects	2	5
Affirmed complementary	Interviewee affirmed complementary use of practices for preventing defects	1	1
Undecided	Interviewee is undecided about a practice to prevent defects	3	5
SW Unit Testing	Assessment of practices for identifying defects at SW Unit Testing stage	6	29
Affirmed individual	Affirmed practice for preventing defects	6	18
Affirmed complementary	Interviewee affirmed complementary use of practices for preventing defects	3	4
Undecided	Interviewee is undecided about a practice to prevent defects	3	7

Name	Description	Files	References
Unit Implementation	Assessment of practices for addressing defects at SW Unit Implementation stage	6	51
Affirmed individual	Affirmed practice for preventing defects	6	38
Affirmed complementary	Interviewee affirmed complementary use of practices for preventing defects	5	11
Undecided	Interviewee is undecided about a practice to prevent defects	2	2