Metadata of the chapter that will be visualized in SpringerLink

Book Title	Systems, Software and Services Process Improvement	
Series Title		
Chapter Title	An Agile-Based Framework for Addressing Defects in Medical Device Software Development	
Copyright Year	2024	
Copyright HolderName	The Author(s), under exclusive license to Springer Nature Switzerland AG	
Corresponding Author	Family Name	**Nyirenda**
	Particle	
	Given Name	**Misheck**
	Prefix	
	Suffix	
	Role	
	Division	Regulated Software Research Centre
	Organization	Dundalk Institute of Technology
	Address	Dundalk, Ireland
	Email	misheck.nyirenda@dkit.ie
	ORCID	http://orcid.org/0000-0003-1320-2755
Author	Family Name	**McHugh**
	Particle	
	Given Name	**Martin**
	Prefix	
	Suffix	
	Role	
	Division	Regulated Software Research Centre
	Organization	Dundalk Institute of Technology
	Address	Dundalk, Ireland
	Email	martin.mchugh@dkit.ie
	ORCID	http://orcid.org/0000-0003-4275-3302
Author	Family Name	**Loughran**
	Particle	
	Given Name	**Róisín**
	Prefix	
	Suffix	
	Role	
	Division	Regulated Software Research Centre
	Organization	Dundalk Institute of Technology
	Address	Dundalk, Ireland
	Email	roisin.loughran@dkit.ie
	ORCID	http://orcid.org/0000-0002-0974-7106
Author	Family Name	**McCaffery**
	Particle	

Given Name	**Fergal**
Prefix	
Suffix	
Role	
Division	Regulated Software Research Centre
Organization	Dundalk Institute of Technology
Address	Dundalk, Ireland
Email	fergal.mccaffery@dkit.ie
ORCID	http://orcid.org/0000-0002-0839-8362

Abstract

Defects in Medical Device Software (MDS) have the potential to cause harm to both patients and caregivers. Research has revealed that defect prevention is often neglected or inadequately implemented in many software development projects. In MDS development, the focus is on defect identification in later stages, typically during coding and testing stages. A recent survey revealed that although MDS development organisations plan to be proactive in defect management by preventing them in early development stages, in practice, they emphasise defect identification in later stages. This approach potentially leads to costly rework and increased risk of defects slipping into the final software release. When using the V-Model, a commonly adopted methodology for safety-critical software development, defect prevention occurs in the early stages on the left side, while defect identification occurs in the later stages on the right side. Studies have revealed that many defects that occur in software can be traced back to the early stages of the development lifecycle. Agile practices provide the potential to prevent defects in the early development stages and identify those that may slip into the later stages. This paper presents an Agile-based Defect Addressing Framework (AbDAF) that is designed to assist MDS development organisations to address defects during the development lifecycle. This framework uses agile practices to address defects by preventing them early on and identifying those that may arise in later stages of development.

Keywords (separated by '-')

Agile Practices - Software Defects - Medical Device Software - Defect Prevention - Defect Identification - Software Quality - Software Safety

An Agile-Based Framework for Addressing Defects in Medical Device Software Development

Misheck Nyirenda(✉), Martin McHugh, Róisín Loughran, and Fergal McCaffery

Regulated Software Research Centre, Dundalk Institute of Technology, Dundalk, Ireland
{misheck.nyirenda,martin.mchugh,roisin.loughran,
fergal.mccaffery}@dkit.ie

Abstract. Defects in Medical Device Software (MDS) have the potential to cause harm to both patients and caregivers. Research has revealed that defect prevention is often neglected or inadequately implemented in many software development projects. In MDS development, the focus is on defect identification in later stages, typically during coding and testing stages. A recent survey revealed that although MDS development organisations plan to be proactive in defect management by preventing them in early development stages, in practice, they emphasise defect identification in later stages. This approach potentially leads to costly rework and increased risk of defects slipping into the final software release. When using the V-Model, a commonly adopted methodology for safety-critical software development, defect prevention occurs in the early stages on the left side, while defect identification occurs in the later stages on the right side. Studies have revealed that many defects that occur in software can be traced back to the early stages of the development lifecycle. Agile practices provide the potential to prevent defects in the early development stages and identify those that may slip into the later stages. This paper presents an Agile-based Defect Addressing Framework (AbDAF) that is designed to assist MDS development organisations to address defects during the development lifecycle. This framework uses agile practices to address defects by preventing them early on and identifying those that may arise in later stages of development.

Keywords: Agile Practices · Software Defects · Medical Device Software · Defect Prevention · Defect Identification · Software Quality · Software Safety

1 Introduction

Medical Device Software (MDS) is increasingly playing an important role in our lives. However, defects in MDS can potentially cause harm to patients and caregivers [1]. Such incidents may ultimately result in significant financial losses and damage to the reputation of MDS manufacturers [2, 3]. Consequently, 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

M. Yilmaz et al. (Eds.): EuroSPI 2024, CCIS 2180, pp. 1–15, 2024.
https://doi.org/10.1007/978-3-031-71142-8_20

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 [4, 5]. 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 [6].

Despite the widely advocated benefit of producing high-quality software through defect prevention in the early stages of software development lifecycle [7], many software development projects often neglect or inadequately implement this important step [8]. Our recent survey reveals a significant emphasis on defect identification in the later stages of MDS development, specifically during coding, testing, release, and maintenance [9].

Considering the increasing complexity of MDS to support a growing number of functionalities, prioritising defect identification while neglecting defect prevention, may increase the risk of defects slipping through to the final release of the software [10]. This arises due to the increased likelihood of developers missing the critical variables associated with the medical environment [11]. The growing complexity of MDS poses significant challenges in ensuring its safety. Therefore, it is essential for MDS organisations to implement a balanced approach of defect prevention and identification throughout the development lifecycle, or to prioritise and practice defect prevention during the early development stages.

In light of the potential harm that defects in MDS can cause to both patients and caregivers and the associated economic losses, the saying that "prevention is better than cure" should be applied throughout the development lifecycle of MDS. More importantly, prioritising defect prevention during the early development stages is vital. This is especially cognizant of the fact that the longer a defect lingers in the development lifecycle, the more costly it becomes to find and fix in later stages [12].

Agile practices provide the potential to prevent defects in the early stages of the development lifecycle and identify those that might slip through to the later stages [10]. This paper presents an Agile-based Defect Addressing Framework (AbDAF) that is designed to assist MDS development organisations to address defects during the development lifecycle. AbDAF uses agile practices identified through a literature review [10] and a survey of MDS organisations in Ireland. The focus is on addressing defects in MDS by preventing them early on and identifying those that may arise in later stages of development. The rest of the paper is organized as follows. Section 2 provides some background to the research. Section 3 presents the survey and results. Section 4 presents an Agile-based Defect Addressing Framework. Section 5 provides a conclusion to the paper and outlines next steps in our research.

2 Impact of Defects in Medical Device Software

MDS has enhanced the capabilities of medical devices, enabling advanced functions like tracking patients' vital signs, alerting physicians to the risk of adverse drug events, and regulating medication dosage for patients [13]. MDS has also caused medical devices failures, leading to injuries and deaths of patients, and numerous recalls of these devices [1, 10]. For instance, in the US, from January 2018 to June 2022, 10.6% of 189 Class I medical devices recalls were due to software defects [14]. 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 [15, 16]. From January 2023 to December 2023, 56 medical devices were recalled due to software defects [17]. 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). These issues underscore the severity of defects in MDS development. It is vital to implement techniques that can assist to address defects early in the development lifecycle of MDS to avoid such consequences.

The objective of our research is to assist MDS development organisations to address defects by using agile practices. In a previous study we reported on the identification of 25 agile practices through a literature review, and a further review and analysis of each of the 25 agile practices [10]. Our aim was to identify agile practices that can assist in addressing defects when developing MDS. The study recommended 17 agile practices for addressing defects in MDS development as follows: Pair Programming (PP), Test-Driven Development (TDD), Continuous Integration (CI), Unit Testing, Refactoring, Integration Testing, Onsite Customer, Code Review, User Acceptance Testing (UAT), Code Ownership, Coding Standards, Small Releases, User Story, Simple Design, Use Case, Behaviour-Driven Development (BDD), and Model Storming. Additionally, the study revealed that agile practices are generally adopted in a plan driven methodology to accommodate changing requirements and to reduce development time. The study also revealed that the V-Model is a common choice for MDS development due to its capability to produce key deliverables needed for regulatory approval. Furthermore, the research found only one study that reported using an agile practice to specifically address defects during MDS development [18]. This study utilized onsite customer or user involvement through methods like user co-design and participatory design and reported significant reduction in defects.

To gain firsthand information and insight into which agile practices are currently used and how they are applied for addressing defects in MDS development, we conducted a survey of developers in MDS organisations in Ireland. The survey and results are presented in Sect. 3.

2.1 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 [19, 20]. This process usually happens in the early stages of the software development lifecycle, including requirements gathering and analysis, system design, architectural design, module design, and implementation [7, 21]. While defect prevention is crucial in software development, it is often overlooked or improperly executed using ineffective methods [8]. Defect identification is the process of detecting defects in software modules with the goal of enhancing the quality of a software system [22]. This process occurs in the later stages of the software development lifecycle, including implementation, testing, release, and maintenance [7, 21]. These processes are illustrated in Fig. 1 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 [23–25].

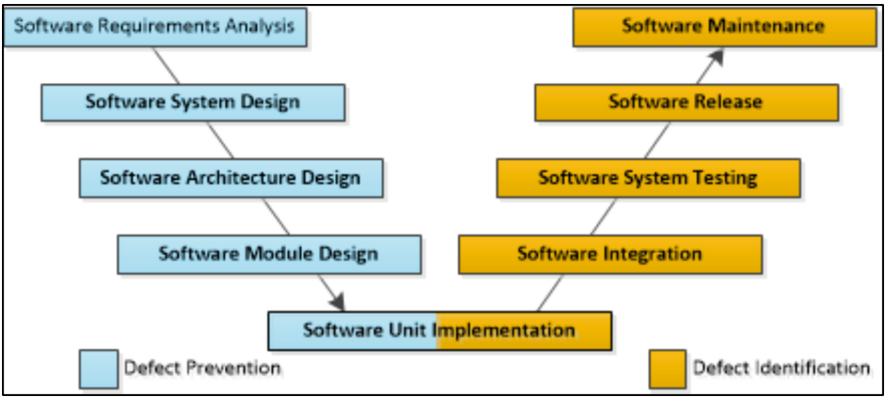

Fig. 1. Defect Prevention and Defect Identification

3 Survey of Medical Device Software Organisations in Ireland

3.1 Survey Process

The Irish Medical Devices Association (2022–2025) reveals that there are over 450 MedTech organisations in Ireland [26]. However, they do not provide a detailed list of these organisations. In contrast, the 2016–2020 strategy categorises several MedTech organisations including 62 that develop MDS [27]. We used the 2016–2020 strategy document as a starting point in identifying MDS organisations that were contacted for the survey. We also invited MDS organisations with existing working relationships with the Regulated Software Research Centre to participate in the survey. Subsequently, an information leaflet, a consent form, and a link to the survey were sent through email to 65 developers from different MDS development organisations. As per the European Union's definition of Small and Medium-sized Enterprises (SMEs) [28], we received responses from 15 different organisations that included 12 SMEs and 3 large organisations.

3.2 Survey Results and Discussion

The survey revealed varied experience in MDS development among the organisations ranging from 1 year to over 20 years. The survey also revealed that 60% of organisations, including 7 SMEs and 2 large organisations, use exclusively an agile methodology 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 hybrid methodology which combines traditional and agile approaches.

Of the 60% that use exclusively an agile methodology, 43% SMEs have used it for about 5 years and the remainder for about 15 years. One of the 2 large organisations has used an agile methodology for less than 5 years while 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% SMEs that use both the V-Model and Waterfall have used these for about 20 years. Of the 27% that use a hybrid methodology, 33% SMEs have used it for over 20 years

and the remainder for less than 5 years. The remaining large organisation from the 27% 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.

This survey also revealed that most organisations address defects during Testing (93%), Coding (87%), Release (80%), and Maintenance (73%). The data analysis also revealed that 87% of the organisations use the agile practices shown in Table 1 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 specifically during coding and testing stages as shown in Table 2.

Table 1. Agile practices usage for defects in MDS development

Agile practice	Use for 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%

When asked to provide comments about using agile practices to address defects, two respondents from SMEs that use agile practices for addressing defects provided contrasting perspectives regarding agile adoption in MDS development. One expressed concern that agile might lack the aspect of feedback from clients, especially those that the organisation has trusted relationship with. The respondent stated as follows,

Agile misses aspects of feedback from key customers where you have trusted relationships to test new features/Alpha versions, etc. and give feedback.

The other respondent suggested that misconceptions that agile practices are not suitable for MDS development might be hindering their adoption and use for addressing defects, stating:

I believe that a lack of understanding and alignment on what is meant by agile, particularly outside development teams, leads to a perception that agile is not a good fit for medical device software development. My view is that there are some core principles of agile practices that promote good software development and reduce the number of defects and help ensure the software is delivering on the user needs.

These contrasting views may potentially explain the slow adoption of agile and the limited use of agile practices particularly for addressing defects in MDS development, as indicated in the existing literature [10]. The view about the lack of understanding for those not part of the development teams may suggest that MDS development teams are eager to adopt agile practices but that they may be hindered by the lack of support from senior management [29]. This could be because of management's fear of risking regulatory approval already obtained [30]. This situation may suggest that any strategy regarding agile implementation in MDS development should involve all key stakeholders including management.

Table 2. 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%

Contrary to the literature suggesting the Waterfall and V-Model as the commonly used for MDS development, the survey results above reveal 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 [30]. The shift towards agile practices can be attributed to the extensive research over the years that identified and addressed the challenges of using agile practices for MDS development [30].

The survey results also reveal that when addressing defects in MDS development, emphasis is on defect identification. As shown in Table 2, 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, defects prevention is often overlooked or inadequately implemented using inefficient methods [8]. Prioritising defect identification while neglecting defect prevention can lead to costly consequences [20].

Although there is similarity in the agile practices identified from the literature review [10] and those from the survey (Table 1), an observable difference exists in the agile practices commonly used for addressing defects. The literature review study found 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 reveals 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, solely relying on defect identification can lead to costly rework and a higher risk of the final MDS failure.

4 An Agile Defect-Based Framework

As discussed in earlier sections, neglecting defect prevention in MDS development can lead to costly rework and even software failure. It is important in MDS development to adopt a holistic approach that prioritises defect prevention and is complemented by defect identification. Agile practices can assist to address defects in MDS development. The insights gathered from both the literature review study and the survey were instrumental in devising a solution for addressing defects in MDS development. The research led to the development of an Agile-based Defect Addressing Framework (AbDAF) that is presented in this paper. AbDAF 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. Figure 2 shows the AbDAF. The framework aligns with the IEC62304 – Medical Device Software – Software life-cycle processes [31] and development methodologies from the survey results above and literature review findings. As shown in Fig. 2, 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 framework emphasises involvement of the customer or proxy in the early development stages to prevent requirements defects and design defects early on [32]. This reduces unnecessary costs that would be incurred when these defects arise in later stages.

Agile practices shown in blue in Fig. 2 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 may be selected as required. Additionally, where a number of agile practices, or all, are selected to be implemented in 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.

As shown in Fig. 2, AbDAF recommends a continuous and iterative process at each stage and provides the flexibility to correct a defect from any previous stage. 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 correction at the software architectural design stage, it will be corrected at the software architectural design stage before proceeding to the implementation stage. Similarly, a defect detected during software unit implementation stage requiring significant modifications at the requirements analysis stage, it will be corrected at the requirements analysis stage before progressing to the software unit testing stage.

At the early stages of many software development projects, clients are often not clear about their system requirements [33]. 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 [34]. 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 [35]. 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 [36]. The following sections discuss the implementation of agile practices in the framework.

4.1 Requirements Gathering Stage

At this stage, AbDAF 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 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.

4.2 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 [37, 38]. This ensures that such defects do not arise in later development stages where they become costly to find and fix [39]. 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. AbDAF 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 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 alternate writing and reviewing 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.

4.3 Software Architectural Design Stage

An important output of this stage is the software architecture that shapes the overall vision of the software product [40]. 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 [41]. 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, AbDAF recommends applying Model Storming in combination with Onsite Customer followed by a combination of Simple Design and Onsite Customer 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. When an understanding is reached between architects and 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 [42]. Additionally, AbDAF recommends that Integration Testing should be an integral part of the entire architectural design process to resolve potential integration defects early. 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 in the interrelations among the components.

4.4 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, AbDAF recommends applying a combination of Model Storming, Onsite Customer, and Integration Testing followed by a combination of Simple Design, Onsite Customer, and Integration Testing. 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. Similar to the preceding stage, Integration Testing should be an integral part of this stage to catch potential integration defects early. This structured application of these agile practices 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.

4.5 Software Unit Implementation Stage

The desirable output at this stage is defect-free and efficient code that accurately accomplishes the functionality stated in the requirement. 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. AbDAF 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, loping 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 proceeding to the next stage.

4.6 Software Unit Testing Stage

At this stage, AbDAF 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 corrected

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.

4.7 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, AbDAF recommends using a combination of Integration Testing, Pair Programming, Refactoring, and Unit Testing followed by Onsite Customer and UAT to review and verify the integrated system. This approach ensures that defective modules or subsystems are identified, refactored and retested. 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.

4.8 Software System Testing Stage

At this stage, AbDAF recommends using User Acceptance Testing, Pair Programming, 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 back 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.

4.9 Acceptance Testing Stage

At this stage, AbDAF recommends applying User Acceptance Testing, Onsite Customer, and Pair Programming. Defects identified through User Acceptance Testing 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 implementation of MDS. Defects that escape to the final release of MDS can lead to devastating consequences.

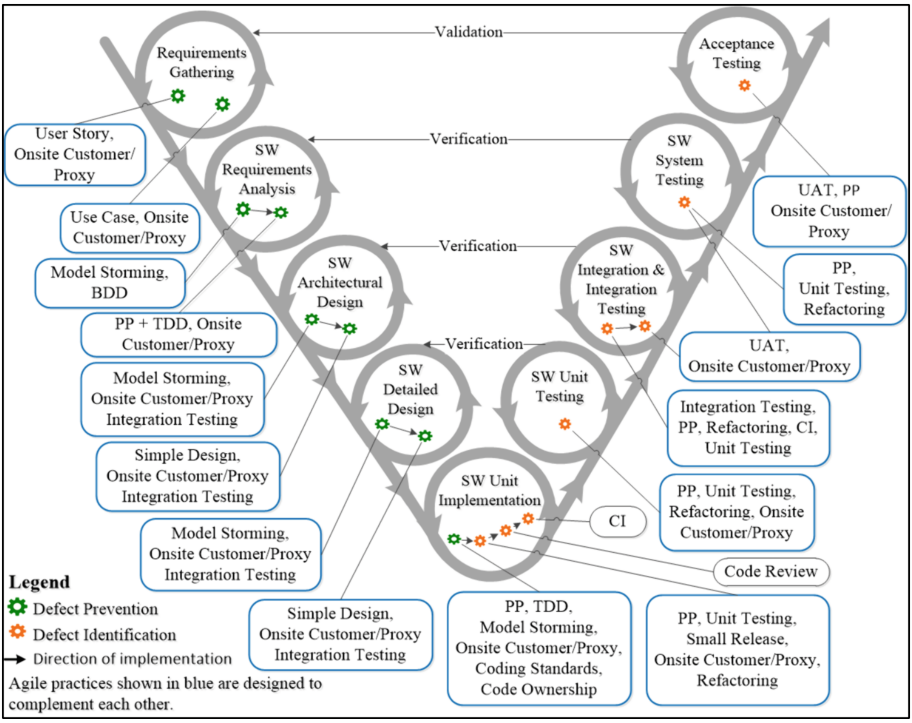

Fig. 2. Agile-based Defect Addressing Framework

5 Conclusion and Future Work

Ensuring the safety of MDS should be an essential consideration throughout the development lifecycle due to the potential risks that MDS defects can cause to both patients and caregivers. However, our survey reveals that MDS development organisations in Ireland emphasise on defect identification in later stages of the development lifecycle. This approach can lead to costly rework and even the ultimate failure of final software which can result in catastrophic consequences. Adopting a holistic approach that prioritises defect prevention and is complemented by defect identification is essential in MDS development. Agile practices provide the potential to prevent defects in the early development stages and identify those that may slip into the later stages. This paper has presented AbDAF which has been designed to assist MDS development organisations to address defects during development. AbDAF uses agile practices to address defects by preventing them early on and identifying those that may arise in later stages of development. Agile practices used in the framework were identified through a literature review study and a survey reported in this paper. By implementing this framework, MDS development organisations can reduce the risk of defects and avert reputation damage and financial loss. As part of our future work, we will experiment AbDAF in an MDS development organisation to evaluate its efficacy. The feedback received from the evaluation will be used to improve the framework.

Acknowledgments. This research is funded through the HEA Landscape and Technological University Transformation Fund, co-funded by Dundalk Institute of Technology.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Leveson, N.G.: The therac-25: 30 years later. Computer **50**, 8–11 (2017). https://doi.org/10.1109/MC.2017.4041349
2. Hovorushchenko, T., Popov, P.: Method of developing the defect-free medical software by establishing the presence of residual defects. In: 4th International Conference on Informatics & Data-Driven Medicine (2021)
3. Krasner, H.: The cost of poor software quality in the US: a 2020 report (2020)
4. Ahmed Khalid, T., Yeoh, E.-T.: Enhancing software development cost control by forecasting the cost of rework: preliminary study. IJEECS **21**, 524 (2021). https://doi.org/10.11591/ijeecs.v21.i1.pp524-537
5. Kotagi, V., Yadav, S.K.: Defect analysis in requirements testing stage of software development life cycle (2023). https://papers.ssrn.com/abstract=4520332. https://doi.org/10.2139/ssrn.4520332
6. Li, H., Liu, Y., Qi, X., Yu, X., Guo, S.: Structuring meaningful bug-fixing patches to fix software defect. IET Software **17**, 566–581 (2023). https://doi.org/10.1049/sfw2.12140
7. Chakravarty, K., Singh, J.: Optimizing defect removal efficiency by defect prediction using machine learning. In: 2022 OITS International Conference on Information Technology (OCIT), pp. 205–210. IEEE, Bhubaneswar, India (2022). https://doi.org/10.1109/OCIT56763.2022.00047
8. Jamal, N., Zulqarnain, M., Boota, M.W., Khan, S., Akber, S.M.A.: Role of defect prevention techniques vs defect detection to improve software quality: critical analysis summary of defect preventive approaches, vol. 2 (2015)
9. Nyirenda, M., McHugh, M., Loughran, R., McCaffery, F.: Using agile practices to address defects in medical device software development: a survey of medical device software organisations in Ireland. In: 8th International Conference on Computer, Software and Modeling. (Accepted for publication), Paris (2024)
10. Nyirenda, M., Loughran, R., McHugh, M., Nugent, C., McCaffery, F.: Identifying agile practices to reduce defects in medical device software development. In: Yilmaz, M., Clarke, P., Riel, A., Messnarz, R. (eds.) Systems, Software and Services Process Improvement. pp. 61–75. Springer Nature Switzerland, Cham (2023). https://doi.org/10.1007/978-3-031-42310-9_5
11. Critical Software: Role of Software in Medical Device Failures | WhitePaper (2024). https://criticalsoftware.com/en/resource/software-in-medical-device-failures. Accessed 5 March 2024
12. Felix, E.A., Lee, S.P.: Predicting the number of defects in a new software version. PLoS ONE **15**, e0229131 (2020). https://doi.org/10.1371/journal.pone.0229131
13. Ronquillo, J.G., Zuckerman, D.M.: Software-related recalls of health information technology and other medical devices: implications for FDA regulation of digital health: recalls: implications for FDA regulation of digital health. Milbank Q. **95**, 535–553 (2017). https://doi.org/10.1111/1468-0009.12278
14. Mooghali, M., Ross, J.S., Kadakia, K.T., Dhruva, S.S.: 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. MDER. **16**, 111–122 (2023). https://doi.org/10.2147/MDER.S412802

15. FDA: 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 (2022)
16. FDA: Vyaire Medical Recalls bellavista 1000 and 1000e Series Ventilators Due to Issues with Software Configurations. FDA (2022)
17. FDA: Medical Device Recalls form January 2023 to July 2023. 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
18. Tang, T., Lim, M.E., Mansfield, E., McLachlan, A., Quan, S.D.: Clinician user involvement in the real world: designing an electronic tool to improve interprofessional communication and collaboration in a hospital setting. Int. J. Med. Inform. **110**, 90–97 (2018). https://doi.org/10.1016/j.ijmedinf.2017.11.011
19. Memon, M.A., Baloch, M.-U.-R.M., Memon, M., Musavi, S.H.A.: Defects prediction and prevention approaches for quality software development. Int. J. Adv. Comput. Sci. Appl. **9** (2018). https://doi.org/10.14569/IJACSA.2018.090857
20. Huang, F., Liu, B.: Software defect prevention based on human error theories. Chin. J. Aeronaut. **30**, 1054–1070 (2017). https://doi.org/10.1016/j.cja.2017.03.005
21. Ergasheva, S., Kruglov, A.: Software development life cycle early stages and quality metrics: a systematic literature review. J. Phys. Conf. Ser. **1694**, 012007 (2020). https://doi.org/10.1088/1742-6596/1694/1/012007
22. Mahmoud, A.N., Santos, V.: Statistical analysis for revealing defects in software projects: systematic literature review. IJACSA **12** (2021). https://doi.org/10.14569/IJACSA.2021.0121128
23. Hauschild, A.-C., Martin, R., Holst, S.C., Wienbeck, J., Heider, D.: Guideline for software life cycle in health informatics. iScience **25**, 105534 (2022). https://doi.org/10.1016/j.isci.2022.105534
24. Liu, B., Zhang, H., Zhu, S.: An incremental V-model process for automotive development. In: 2016 23rd Asia-Pacific Software Engineering Conference (APSEC), pp. 225–232. IEEE, Hamilton, New Zealand (2016). https://doi.org/10.1109/APSEC.2016.040
25. McHugh, M., McCaffery, F., Coady, G.: An agile implementation within a medical device software organisation. In: Mitasiunas, A., Rout, T., O'Connor, R.V., Dorling, A. (eds.) Software Process Improvement and Capability Determination. pp. 190–201. Springer International Publishing, Cham (2014). https://doi.org/10.1007/978-3-319-13036-1_17
26. Ibec: Medtech Strategy 2025 – IBEC. https://www.ibec.ie/connect-and-learn/industries/life-sciences-and-healthcare/medtech-strategy-2025. Accessed 27 Nov 2023
27. Ibec: Irish Medtech 2020 Strategy The Global Hub (2016). 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
28. EU: The new SME definition: User guide and model declaration (2022). https://www.eusmecentre.org.cn/wp-content/uploads/2022/12/SME-Definition.pdf
29. Campanelli, A.S., Bassi, D., Parreiras, F.S.: Agile transformation success factors: a practitioner's survey. In: Dubois, E. and Pohl, K. (eds.) Advanced Information Systems Engineering, pp. 364–379. Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-59536-8_23
30. McHugh, M., McCaffery, F., Casey, V.: Barriers to adopting agile practices when developing medical device software. In: Mas, A., Mesquida, A., Rout, T., O'Connor, R.V., Dorling, A. (eds.) Software Process Improvement and Capability Determination, pp. 141–147. Springer Berlin Heidelberg, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30439-2_13

31. AAMI/IEC: BS EN 62304–2006+A1:2015 Medical Device Software - Software life-cycle processes, (2015)
32. Alvertis, I., et al.: User involvement in software development processes. Procedia Comput. Sci. **97**, 73–83 (2016). https://doi.org/10.1016/j.procs.2016.08.282
33. Haleem, M., Farooqui, M.F., Faisal, M.: Tackling Requirements uncertainty in software projects: a cognitive approach. Int. J. Cogn. Comput. Eng. **2**, 180–190 (2021). https://doi.org/10.1016/j.ijcce.2021.10.003
34. Suma, V., Nair, T.R.: Effectiveness of defect prevention in I.T. for product development (2010). https://doi.org/10.48550/arXiv.1001.3725
35. Michael, K.A., Boniface, K.A.: Inadequate requirements engineering process: a key factor for poor software development in developing nations: a case study, vol. 8 (2014)
36. Kamalrudin, M., Ow, L.L., Sidek, S.: Requirements defects techniques in requirements analysis: a review, vol. 10 (2018)
37. Alshazly, A.A., Elfatatry, A.M., Abougabal, M.S.: Detecting defects in software requirements specification. Alex. Eng. J. **53**, 513–527 (2014). https://doi.org/10.1016/j.aej.2014.06.001
38. Margarido, I.L., Faria, J.P., Vidal, M., Vieira, M.: Classification of defect types in requirements specifications: literature review, proposal and assessment. In: Information Systems and Technologies (CISTI) (2011)
39. Basak, S., Shazzad Hosain, M.: Software testing process model from requirement analysis to maintenance. IJCA **107**, 14–22 (2014). https://doi.org/10.5120/18795-0147
40. Dasanayake, S., Aaramaa, S., Markkula, J., Oivo, M.: Impact of requirements volatility on software architecture: how do software teams keep up with ever-changing requirements? J Software Evolu Process. **31**, e2160 (2019). https://doi.org/10.1002/smr.2160
41. Kouroshfar, E., Mirakhorli, M., Bagheri, H., Xiao, L., Malek, S., Cai, Y.: 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, pp. 246–257. IEEE, Florence, Italy (2015). https://doi.org/10.1109/MSR.2015.30
42. Wan, Z., Zhang, Y., Xia, X., Jiang, Y., Lo, D.: 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, pp. 1457–1469. ACM, San Francisco CA USA (2023). https://doi.org/10.1145/3611643.3616367

Author Queries

Chapter 20

Query Refs.	Details Required	Author's response
AQ1	This is to inform you that corresponding author has been identified as per the information available in the Copyright form.	