Metadata of the chapter that will be visualized in SpringerLink

Book Title	Systems, Software and Services Process Improvement	
Series Title		
Chapter Title	Identifying Agile Practices to Reduce Defects in Medical Device Software Development	
Copyright Year	2023	
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	**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	**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	**Nugent**
	Particle	

	Given Name	**Christopher**
	Prefix	
	Suffix	
	Role	
	Division	
	Organization	Ulster University
	Address	Newtownabbey, UK
	Email	cd.nugent@ulster.ac.uk
	ORCID	http://orcid.org/0000-0003-0882-7902
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	Medical Device Software (MDS) defects have caused death of patients and continue to be the major cause of recalls of medical devices in the US and Europe. Despite various approaches proposed to address defects, dealing with defects in MDS is an increasingly difficult task as MDS has become more complex to support a growing number of functions. To increase quality in any software development project, it is essential that defects are identified and addressed quickly in the early stages of the software development life cycle. Agile methods have been advocated to increase software quality by minimising defects through their agile practices. However, agile methods on their own are deficient in satisfying the regulatory requirements for the MDS domain. Instead, the common approach is to integrate agile practices into the plan driven methods. Consequently, frameworks have been developed to help developers in the MDS domain to accrue the benefits of agile development while fulfilling regulatory requirements. Despite the adoption of agile practices in MDS development, it is still unclear as to which agile practice(s) is effective and how it is applied to address MDS defects. The purpose of this research is to identify agile practices that can assist in addressing defects in MDS development. This will help MDS developers to select the appropriate agile practice(s) to address defects.

Keywords (separated by '-')	Medical Device Software - Agile Practices - Medical Device Software Defects - Software Faults - Medical Device Recalls

Identifying Agile Practices to Reduce Defects in Medical Device Software Development

Misheck Nyirenda[1](✉), Róisín Loughran[1], Martin McHugh[1], Christopher Nugent[2], and Fergal McCaffery[1]

[1] Regulated Software Research Centre, Dundalk Institute of Technology, Dundalk, Ireland
{misheck.nyirenda,roisin.loughran,martin.mchugh,
fergal.mccaffery}@dkit.ie
[2] Ulster University, Newtownabbey, UK
cd.nugent@ulster.ac.uk

Abstract. Medical Device Software (MDS) defects have caused death of patients and continue to be the major cause of recalls of medical devices in the US and Europe. Despite various approaches proposed to address defects, dealing with defects in MDS is an increasingly difficult task as MDS has become more complex to support a growing number of functions. To increase quality in any software development project, it is essential that defects are identified and addressed quickly in the early stages of the software development life cycle. Agile methods have been advocated to increase software quality by minimising defects through their agile practices. However, agile methods on their own are deficient in satisfying the regulatory requirements for the MDS domain. Instead, the common approach is to integrate agile practices into the plan driven methods. Consequently, frameworks have been developed to help developers in the MDS domain to accrue the benefits of agile development while fulfilling regulatory requirements. Despite the adoption of agile practices in MDS development, it is still unclear as to which agile practice(s) is effective and how it is applied to address MDS defects. The purpose of this research is to identify agile practices that can assist in addressing defects in MDS development. This will help MDS developers to select the appropriate agile practice(s) to address defects.

Keywords: Medical Device Software · Agile Practices · Medical Device Software Defects · Software Faults · Medical Device Recalls

1 Introduction

In the medical domain, software is used to diagnose illness and assist health personnel to perform numerous activities related to patient health. Software used in the medical domain and that performs its functions as a medical device is termed as Medical Device Software (MDS) [1, 2]. MDS allows medical devices to accomplish complex functionalities that would otherwise not be possible. For example, MDS allows medical devices to be utilized in diverse applications, such as regulating the amount of medication that patients receive, tracking patients' vital signs, and alerting caregivers about

M. Yilmaz et al. (Eds.): EuroSPI 2023, CCIS 1891, pp. 1–15, 2023.
https://doi.org/10.1007/978-3-031-42310-9_5

harmful drug incidences [3]. It is particularly vital that MDS must be of high quality and reliable as any software quality issues can have significant detrimental effects on a patient's recovery, health, and well-being [4]. The difficulty in developing reliable and safe MDS is evidenced in the recalls of numerous medical devices by the Food and Drug Administration (FDA) due to software defects [5]. Reliability mainly considers three characteristics: fault avoidance, fault removal, and fault tolerance [6]. It is very difficult to achieve reliable software considering that all software development techniques have limitations, and none can guarantee overall software reliability [7].

Delivering high quality software is a key element in MDS domain. However, some MDS is released with defects which only manifest when the software is used. There are many factors that can cause systems to fail including hardware failures, and software failures [8]. Software-related failures could be traced to different phases of the software development lifecycle.

The purpose of this research is to identify agile practices that could help to address defects in MDS development. We reviewed each of the agile practices individually to gather evidence of their use and potential to address defects in the General Software Domain (GSD). We reviewed studies in the MDS domain to identify which agile practices were used in MDS development, including those used to address defects.

The aim of this study is to help practitioners to select and apply the most suitable agile practice(s) to address defects in MDS development. The remainder of the paper is organized as follows, Sect. 2 covers the background and the state of the art to software defect identification, Sect. 3 covers agile practices used in MDS development and agile practices that could be used to address defects in MDS development, Sect. 4 covers future work and concludes the paper.

2 Background of Medical Device Software Development

2.1 Medical Device Software

In healthcare, the increase in diseases prevalence and shortage of caregivers has necessitated the drive for faster and more innovative technological solutions to save human life [9]. Advances in computing, networking, sensing and medical technology have led to the dramatic increase in diagnostic and therapeutic devices in healthcare [9]. However, the lack of proper integration and operation of these systems presented systematic inefficiencies in healthcare delivery [9]. Consequently, we have witnessed the rise in development and use of software that integrated these different healthcare systems and enabled them to interoperate. Most importantly, many medical devices are software-driven which has enabled them to perform sophisticated functions. Moreover, software used in healthcare can be recognized as a medical device on its own [10].

There are two types of MDS used in the medical domain: Software as a Medical Device (SaMD) also called standalone software, and Software in a Medical Device (SiMD) also known as embedded medical device software [11]. SaMD is "software intended to be used for medical functions that performs its objectives without being part of a hardware medical device" [1, 2]. On the other hand, SiMD describes a "traditional medical device that uses software to support its functionality" [11]. Moreover, the definition of SaMD by the FDA clearly specifies that "software running on a hardware

medical device qualifies as SaMD only if it does not drive or control the medical device" [12].

2.2 Software-Related Medical Device Failures

Despite the availability of numerous standards related to MDS, there have been many recalls of medical devices and medical device failures due to software defects. In 2021, in the US, software issues remained the top cause of medical device failures accounting for 162 (19.4%) medical device recalls [13]. In the EU, software was the top cause of medical device recalls in quarter three and quarter four of 2021 and was the second top reason for all medical device recalls in 2021, accounting for 408 (19.8%) medical device recalls [14]. In 2017 software was the primary cause of medical device failures with "one in every three medical devices being recalled because of software faults" [15]. Analysis of FDA recall data from 1999 to 2010 identified that four out of every ten medical devices comprising software failed due to software defects [15] and from 2014 to 2016 there were 100 software-related recalls [16].

Similar studies [3, 5] also present types of software defects that occurred in medical devices such as control flow faults and omission faults. These and many other software defects could be traced back to the software development lifecycle. Finding and fixing defects quickly in the early stages of software development is less expensive than finding and fixing defects when the software is delivered [17].

2.3 Standards

MDS must be developed in accordance to national and international regulations [18]. Regulatory bodies ensure that medical products do not pose any harm to patients and healthcare personnel. Accordingly, MDS manufacturers must adhere to numerous regulatory standards to ensure their MDS is safe. Standards are set and enforced by government agencies such as the FDA in the US and the European Commission in the EU. In [19] a detailed background to numerous standards relevant to MDS development is provided. The authors provide valuable information about how different standards came into play and how they relate to ensure that MDS is safe. Among the many standards that are relevant to the MDS domain, IEC 62304:2006+A1:2015 Medical Device – Software Life Cycle Processes is of utmost significance to manufacturers as it provides an internationally recognized standard [20]. ISO 14971 Application of Risk Management to Medical Devices and ISO 13485 Medical Devices – Quality Management Systems – Requirements for Regulatory Purposes are also vital to the development of regulatory compliant MDS [19].

2.4 State-of-the-art for Software Defect Identification

Different techniques such as taxonomy-based testing exist to help manufacturers detect software defects at earlier phases of MDS development. However, many software manufacturers avoid using defect-based testing as it requires a detailed defect taxonomy that is costly to build and difficult to validate [21]. The Association for the Advancement of

Medical Instrumentation (AAMI) developed and published AAMI SW91-2018- Classification of Defects in Health Software in 2018 [21]. SW91 is a taxonomy that provides a common language for the classification of software defects that occur in health software [22].

While defect taxonomies such as SW91 help software manufacturers to know the type of defects that occur in software, they do not assist in removing the defects. Techniques are required to identify and remove the defects. We investigate which agile practices can help to detect and remove software defects in the early stages of the software development lifecycle as defects that slip through these phases are costly to find and fix after product delivery [17].

2.5 Agile Software Development

Agile Software Development (ASD) reduces the risk of developing low quality software by minimizing defects through lightweight methods that emphasise customer collaboration and responsiveness to change [23–27]. ASD shortens delivery time while meeting changing customer needs through fast feedback and flexibility to accommodate rapid changes to requirements [27]. There are many agile methodologies such as Scrum and eXtreme Programming (XP) [28, 29]. Each agile methodology has several practices. For example, Scrum includes sprint and sprint retrospective, while XP involves pair programming and refactoring [29]. Agile practices are used to ensure principles and values of an agile method are implemented [30].

Despite the quality benefit advocated when using agile methodologies, research reveals that using them on their own they are unable to satisfy the regulatory requirements for the development of MDS [27, 31, 32]. To meet regulatory requirements, MDS is typically developed in accordance with the V-Model [18, 33, 34]. However, the V-Model is based upon Royce's Waterfall model which is very rigid in terms of requirements [35]. To overcome this issue, MDS developers have integrated specific agile practices into the V-Model [36, 37]. Despite the integration of agile practices in MDS development projects, it is still unclear as to which agile practice(s) is effective and how they can be applied to address MDS defects.

3 Agile Practices for MDS Development

3.1 Identifying all Potentially Suitable Agile Practices

We reviewed several studies to gather agile practices from the commonly used agile methodologies as indicated in a recent report by Digital.ai Agility, (formerly VersionOne) [38]. A study by [29] provides an extensive review of several agile methodologies that are outlined in the report by digital.ai and discuss their practices. In [39] the authors discuss various agile methodologies and practices they use. These studies formed the foundation for our initial list of agile practices. We also reviewed papers such as [40] which identified 50 agile practices that are used in safety-critical software development and [41] which identified 18 agile practices which they called 'universal practices'. These and other studies such as [42] and [43] were some of the sources of an initial list of the

agile practices which were further reviewed to obtain our final list of agile practices. We considered the impact that the agile practice may have on the requirements, modelling, coding, testing and release stages as defects most often arise during these stages [44]. Our final consolidated list of agile practices is presented in Table 1. It is important to note that while some of these practices precede the invention of agile methodologies, they are still incorporated as agile practices in guidance documents such as AAMI TIR45 [45].

3.2 Identifying Which Practices Have Been Used in the Development of MDS

We reviewed 53 papers from the MDS domain that reported using agile practices in MDS development. Of the 53 papers, 34 were relevant to our study as they clearly indicated the agile practices used. The number of studies that utilised the agile practices in the MDS domain are shown in Column 3 of Table 1. Due to size limitation, the data showing a list of all studies and all the agile practices a study utilised cannot be included in this paper. Nevertheless, the data can be provided upon request.

Our review of articles from the MDS domain showed that Scrum and XP were the primary sources of the adopted agile practices. We also observed that agile practices were generally adopted to reduce delivery time, development costs, and to flexibly accommodate changing customer requirements [36, 37, 46–50]. Although quality improvement is mentioned in some studies, it is not clearly related to defects. Moreover, any quality improvement is attributed to the general adoption of the agile practices utilised. One study reported a 78% reduction in field defects across businesses using the SAFe® framework [48]. However, it is unclear which specific agile practice reduced the defects, because not all projects may need to adopt all the agile practices of the SAFe® framework. It is crucial that developers select the appropriate agile practice(s) to address defects in MDS development. In this regard, we reviewed each of the agile practices in our consolidated list to uncover evidence of their use to address software defects in the GSD. The number of studies that discussed the practice in relation to defects are shown in Table 1 column 2.

As shown in Table 1, despite the high number of studies that used agile practices such as sprint, sprint planning meeting, sprint backlog, sprint review meeting, and retrospective in MDS development, to date there is no study that discusses using any of these practices to address software defects. The same appears for planning game and use case. This may mean that research into these practices in relation to software defects has not been conducted or published. It may also mean that these practices have no direct impact on coding and testing during software development. On the other hand, several studies used TDD, PP, unit testing, refactoring, and continuous integration for MDS development. A high number of studies also used these practices in the GSD to address defects. However, to date there is no study that discusses using any of these practices to specifically address defects in MDS development. The high number of studies that used these practices in the GSD may indicate that they have direct impact on design, coding, and testing in the software development process. Few studies discussed using daily stan-up meeting, user story, coding standards, onsite customer, collective ownership, integration testing, simple design, user acceptance testing (UAT), and small releases in relation to addressing defects. Our review did not find any study that discussed using model storming in relation to defects despite the literature indicating its

potential to resolve requirements defects collaboratively and quickly through creation of models [51, 52]. Although few studies used code review in the MDS domain, it was used in several studies for defects in GSD.

Table 1. Number of articles where agile practices have been reported.

Agile practice	Defects in GSD	MDS	Defects in MDS
Test-Driven Development	38	14	0
Unit Testing	25	15	0
Refactoring	23	10	0
Code Review	19	2	0
Pair Programming	17	12	0
Continuous Integration	10	17	0
Collective Ownership	5	3	0
Coding Standards	5	5	0
Small Releases	4	2	0
Onsite Customer	3	6	0
User Acceptance Testing	3	5	0
Daily Stand-up Meeting	3	15	0
Behaviour Driven Development	3	0	0
Integration Testing	2	4	0
Product Backlog	2	17	0
User Story	2	17	0
Simple Design	1	4	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

3.3 Agile Practices that Could be Used to Identify Defects

We conducted a review of each agile practice shown in Table 1 to identify evidence of their use to address defects. We used several search statements to find relevant studies

in sources such as SpringerLink, IEEE Explore, ACM Library, IEEE Computer Society, ScienceDirect, and Google Scholar. The statements included interchangeable terms defect, fault, bug, error, flaw, failure, and anomaly. For example, for PP the following search statements with different alterations were used:

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

Abstract and conclusion sections of studies that contained any of these terms were read and those that discussed addressing them using agile practices were read thoroughly. Studies that did not discuss addressing any of the terms using agile practices were discounted. Column 2 in Table 1 shows number of studies on the effect of agile practices to reduce defects. Overall, the reviewed articles suggest that agile practices can reduce software defects when applied correctly. We present findings on PP as an example of our review process on the agile practices.

Pair Programming for Addressing Defects. 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 [53–56]. The navigator also thinks of alternatives, intervenes to supply necessary information, and considers strategic design implications to prevent or remove defects early, reducing costs [57–59].

We examined 105 papers and found 17 that discussed PP and software defects. The rest (88 studies) discussed a range of other perspectives regarding PP such as system complexity and programmer expertise [60], development effort [61, 62], knowledge transfer [59], productivity [63] etc. Table 2 summarises the 17 studies. The acronyms '*IND*', '*AC*', and '*IND & AC*' denote that the study used professional developers, students, or both as subjects, respectively.

Table 2. Overview of PP usage for defects.

Study type	Domain			Improvement	No improvement
	IND	*AC*	*IND & AC*		
case study	3	0	2	4	1
experiment	0	5	1	5	1
experience report	1	4	0	5	0
survey	1	0	0	1	0
Total	5	9	3	15	2

As shown in Table 2, 29% of the studies report results from industry, 53% from the academic domain, and 18% from both industry and academic domains. Fifteen of the 17 studies report improvement in using PP to reduce software defects. These include 29% studies from the industry domain of which 3 are case studies [64–66], one is an experience report [67], and one is a survey [68]. A study by [69] found that PP prevents defects and lowers the number of defects in software. They further state that the combination of TDD and designing in pairs effectively lowered the defects. In [65] the authors report lower defect rates for the parts of code where PP was used in comparison to other parts. They concluded that PP helps to reduce the introduction of new defects when existing code is modified. A study by [66] found that an increased amount of PP within tasks lowered the number of defects and reduced the introduction of new defects. In [67] a reduction in error rate that was "three orders of magnitude" than normal for the organisation was achieved after PP was used. A study by [68] found that the number of defects for PP teams was lower than for solo groups.

Four of the 17 studies are academic experiments. A study by [70] found that PP yielded 40% fewer defects than Fagan inspection. Fagan Inspection is a group review method that involves six steps: planning, overview, preparation, inspection, rework, and follow up to detect defects in development documents like specifications, designs, source code, etc., during the various stages of software development process [71]. In [72] a lower defect count for PP teams than solo programmers is reported. A study by [73] found that the defect densities of the PP groups were much lower than those of the traditional programming groups. A study by [74] found higher defect reduction rates during integration for PP team than during inspection for TSP team. Four other studies of the 17 are experience reports from academic domain. A study by [75] report fewer defects for PP groups as compared to second-best solo programmers. A study by [76] found that the defect densities of PP groups were much lower than those of the traditional programming groups. In [77] the defect density for PP group was much lower than for solo group. The authors concluded that PP was much effective in reducing defects than solo programming. A study by [58] reported that programs written by PP groups passed more automated tests, resulting in less defects as compared to solo groups. Two of the 17 studies report results from both industry and academic. One of these is an experiment by [78] who found that PP yielded 40% fewer defects than Fagan inspection. The other study was a case study by [79] who found that PP was beneficial in reducing defects.

Two of the 17 studies did not find significant difference in defect reduction between PP groups and solo groups. One of the two studies is an *IND & AC* case study that examined whether PP improves software quality in four projects [54]. The authors calculated relative defect density in two of the four projects where defects were properly recorded. Their results showed that in one of the projects the relative defect density between pair and solo programming were almost equal, 7.0 defects/KLOC and 6.9 defects/KLOC, respectively. However, a significant difference of 1.3 defects/KLOC for PP and 8.4 defects/KLOC for solo programming was noted in the other project. Thus, the authors could not conclude whether PP lowers the defect density as the results were conflicting. The other study examined PP in relation to thoroughness and defect finding effectiveness of test suites [80]. The result showed that PP did not have significant impact on thoroughness and defect detection effectiveness. However, the author attributed the result to the small size of the project in which branch coverage and mutation score indicator tend

to be insignificant as development skill alone may be enough without requiring a second pair of eyes. Overall, the findings from the reviewed papers on PP appear to suggest that PP can help to address defects in software development if applied correctly. The findings also suggest that when a developer works with a partner, there is higher chance that defects can be detected early as there is always someone examining the code as it is being developed. In general, the positive effects of using PP for reducing defects far outweigh the few negative effects reported.

3.4 Agile Practices that Could Potentially Help to Address Defects in MDS Development

Our review of agile practices has revealed that certain agile practices have the potential to reduce defects in software development. Based on review findings, the impact on various development stages, and the fact that some of the reviewed practices have been adopted and used in MDS development, we recommend agile practices such as TDD, PP, unit testing, refactoring, code review, continuous integration, integration testing, UAT, user story and onsite customer, to help address defects in MDS development. Our recommendation for certain agile practices is also based on their ability to enhance quality as detailed in the literature [81]. Despite limited studies on certain agile practices, the information strongly indicates potential to reduce defects and ensure the right product is developed. For example, UAT 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 [82, 83]. Similarly, user story helps define and understand software requirements correctly [84], enabling developers to build the right software product, avoiding costly defects later in the development life cycle [85]. We recommend applying UAT early 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 become bigger [86]. Coding standards and code ownership reinforce other practices such as refactoring and PP, making maintenance easier and allowing all developers to modify the code. TDD, PP, continuous integration, unit testing, code review, onsite customer, and user story help identify and remove defects early in software development [87].

4 Future Work

As part of our future work, we plan to conduct extensive research through surveys, interviews, and focus groups with industry professionals to identify agile practices that are most effective in addressing defects. Based on the findings from this research and the work presented in this paper, we will develop a comprehensive framework. This framework will incorporate the most effective agile practices for addressing defects and will be aligned with the mapping of two industry standards: SW91 and IEC62304 MDS software development life cycle processes. Our goal is to help MDS developers detect and remove defects early in the development life cycle, thereby preventing costly rework when defects are discovered later during use. By implementing this framework, MDS manufacturers can reduce the risk of defects, avoiding reputation damage and economic loss.

5 Conclusion

In this study we investigated agile practices collected from the commonly used agile methods with the aim of identifying those that could help to address defects in MDS development. We reviewed articles from the MDS domain to identify practices used to develop MDS and if any were used to address defects. We found that agile practices are generally used in MDS development to reduce delivery time, development costs, and to flexibly accommodate changing customer requirements. We reviewed each agile practice individually and found that TDD, PP, refactoring, unit testing, integration testing, continuous integration, code review, onsite customer, user acceptance testing, coding standards, collective ownership, and BDD help to identify and remove defects in the GSD. However, no study has specifically discussed using any agile practice to address defects in MDS development.

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

References

1. Pashkov, V., Gutorova, N., Harkusha, A.: Medical device software: defining key terms. Wiadomości Lekarskie **69**(6), 813–817 (2016)
2. IMDRF: "Software as a Medical Device": Possible Framework for Risk Categorization and Corresponding Considerations 30 (2014)
3. Ronquillo, J.G., Zuckerman, D.M.: Software-related recalls of health information technology and other medical devices: implications for FDA regulation of digital health. Milbank Q. **95**, 535–553 (2017). https://doi.org/10.1111/1468-0009.12278
4. Shroff, V., Reid, L., Richardson, I.: A Proposed Framework for Software Quality in the Healthcare and Medical Industry 8 (2011)
5. Alemzadeh, H., Iyer, R.K., Kalbarczyk, Z., Raman, J.: Analysis of safety-critical computer failures in medical devices. IEEE Secur. Priv. **11**, 14–26 (2013). https://doi.org/10.1109/MSP.2013.49
6. Mili, A., Cukic, B., Xia, T., Ben Ayed, R.: Combining fault avoidance, fault removal and fault tolerance: an integrated model. In: 14th IEEE International Conference on Automated Software Engineering, pp. 137–146. IEEE Comput. Soc, Cocoa Beach, FL, USA (1999). https://doi.org/10.1109/ASE.1999.802168
7. Knight, J.C., Wika, K.G.: Software safety in medical applications. Comput. Aided Surg. **1**, 121–132 (2010). https://doi.org/10.3109/10929089509105686
8. Wallace, D., Kuhn, D.: Failure modes in medical device software: an analysis of 15 years of recall data. Int. J. Reliab. Qual. Saf. Eng. **8**, (2002). https://doi.org/10.1142/S0218539301000058X
9. Lee, I., et al.: High-confidence medical device software and systems. Computer **39**, 33–38 (2006). https://doi.org/10.1109/MC.2006.127
10. Pashkov, V., Harkusha, A.: Stand-alone software as a medical device: qualification and liability issues. Wiad. Lek. Wars. Pol. 1960 **73**, 2282 (2020). https://doi.org/10.36740/WLek202010134
11. Gordon, W., Stern, A.D.: Challenges and opportunities in software-driven medical devices. Nat. Biomed. Eng. **3**, 493–497 (2019). https://doi.org/10.1038/s41551-019-0426-z

12. FDA, C. for D. and R.: Digital Health Criteria. FDA. (2020)
13. Sedgwick Brand Protection: State of the Nation 2022 Recall Index Report (2022)
14. Sedgwick Brand Protection: EU State of the nation 2022 recall index report (2022)
15. Fu, Z., Guo, C., Zhang, Z., Ren, S., Jiang, Y., Sha, L.: Study of software-related causes in the FDA medical device recalls. In: 2017 22nd International Conference on Engineering of Complex Computer Systems (ICECCS), pp. 60–69 (2017). https://doi.org/10.1109/ICECCS.2017.20
16. Bliznakov, Z., Stavrianou, K., Pallikarakis, N.: 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. pp. 1174–1177. Springer International Publishing, Cham (2014). https://doi.org/10.1007/978-3-319-00846-2_291
17. Boehm, B., Basili, V.R.: Software defect reduction top 10 list. Computer **34**, 135–137 (2001). https://doi.org/10.1109/2.962984
18. Kosti, M.D.: Challenges of agile practices implementation in the medical device software development methodologies. Eur. Proj. Manage. J. **7**, 9 (2017)
19. McHugh, M., McCaffery, F., Coady, G.: Adopting agile practices when developing medical device software. Comput. Eng. Inf. Technol. **04**, 14 (2015). https://doi.org/10.4172/2324-9307.1000131
20. ISO: IEC 62304:2006(en), Medical device software — Software life cycle processes, https://www.iso.org/obp/ui/#iso:std:iec:62304:ed-1:v1:en (2006)
21. Rajaram, H.K., Loane, J., MacMahon, S.T., Mc Caffery, F.: Taxonomy-based testing and validation of a new defect classification for health software. J. Softw. Evol. Process. **31**, e1985 (2019). https://doi.org/10.1002/smr.1985
22. ANSI/AAMI: ANSI/AAMI SW91: 2018 Classification of Defects in Health Software (2018)
23. Noor, R., Fahad Khan, M.: Defect management in agile software development. Int. J. Mod. Educ. Comput. Sci. **6**, 55–60 (2014). https://doi.org/10.5815/ijmecs.2014.03.07
24. Abdelaziz, A.A., El-Tahir, Y., Osman, R.: Adaptive software development for developing safety critical software. In: 2015 International Conference on Computing, Control, Networking, Electronics and Embedded Systems Engineering (ICCNEEE), pp. 41–46. IEEE, Khartoum, Sudan (2015). https://doi.org/10.1109/ICCNEEE.2015.7381425
25. Beecham, S., Noll, J., Richardson, I.: Using agile practices to solve global software development problems -- a case study. In: 2014 IEEE International Conference on Global Software Engineeering Workshops, pp. 5–10. IEEE, Shanghai, China (2014). https://doi.org/10.1109/ICGSEW.2014.7
26. Turk, D., Robert, F., Rumpe, B.: Assumptions underlying agile software-development processes. J. Database Manag. **16**, 62–87 (2005). https://doi.org/10.4018/jdm.2005100104
27. Diebold, P.: ACAPI - Agile Capability Analysis and Process Improvement in Highly Regulated Environments. Kaiserslautern (2013)
28. Ibrahim, N.: An overview of agile software development methodology and its relevance to software engineering. Jurnal Sistem Informasi **2**, 12 (2007)
29. Abrahamsson, P., Salo, O., Ronkainen, J.: Agile software development methods: review and analysis 112 (2002)
30. Tripp, J.F., Armstrong, D.J.: Agile methodologies: organizational adoption motives, tailoring, and performance. J. Comput. Inf. Syst. **58**, 170–179 (2018). https://doi.org/10.1080/08874417.2016.1220240
31. McHugh, M., McCaffery, F., Fitzgerald, B., Stol, K.-J., Casey, V., Coady, G.: Balancing agility and discipline in a medical device software organisation. In: Woronowicz, T., Rout, T., O'Connor, R.V., Dorling, A. (eds.) SPICE 2013. CCIS, vol. 349, pp. 199–210. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38833-0_18

32. Myklebust, T., Stålhane, T., Hanssen, G.: Use of agile practices when developing safety-critical software. Presented at the August 9 (2016)
33. 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.) SPICE 2012. CCIS, vol. 290, pp. 141–147. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30439-2_13
34. McHugh, M., McCaffery, F., Casey, V., Pikkarainen, M.: Integrating Agile Practices with a Medical Device Software Development Lifecycle (2012)
35. Özcan-Top, Ö., McCaffery, F.: A hybrid assessment approach for medical device software development companies. J. Softw. Evol. Process. **30**, e1929 (2018). https://doi.org/10.1002/smr.1929
36. McHugh, M., Cawley, O., McCaffcry, F., Richardson, I., Wang, X.: 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 (2013). https://doi.org/10.1109/SEHC.2013.6602471
37. Rasmussen, R., Hughes, T., Jenks, J.R., Skach, J.: Adopting agile in an FDA regulated environment. In: 2009 Agile Conference, pp. 151–155 (2009). https://doi.org/10.1109/AGILE.2009.50
38. Digital.ai Agility: 16th Annual State of Agile Report (2022)
39. de Azevedo Santos, M., de Souza Bermejo, P.H., de Oliveira, M.S., Tonelli, A.O., Seidel, E.J.: Improving the management of cost and scope in software projects using agile practices. Int. J. Comput. Sci. Inf. Technol. **5**, 47–64 (2013). https://doi.org/10.5121/ijcsit.2013.5104
40. Myklebust, T., Lyngby, N., Stålhane, T.: Agile practices when developing safety systems. Los Angel. (2018)
41. Diebold, P., Dahlem, M.: Agile practices in practice: a mapping study. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, pp. 1–10. ACM, London England United Kingdom (2014). https://doi.org/10.1145/2601248.2601254
42. Haynes, S.R., Friedenberg, M.: Best Practices in Agile Software Development (2006)
43. Jain, R., Suman, U.: Effectiveness of agile practices in global software development. Int. J. Grid Distrib. Comput. **9**, 231–248 (2016). https://doi.org/10.14257/ijgdc.2016.9.10.21
44. Kannan, V., et al.: User stories as lightweight requirements for agile clinical decision support development. J. Am. Med. Inform. Assoc. **26**, 1344–1354 (2019). https://doi.org/10.1093/jamia/ocz123
45. AAMI: AAMI TIR45: 2012; Guidance on the use of agile practices in the development of medical device software, (2012). https://webstore.ansi.org/standards/aami/aamitir452012r2018
46. 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.) SPICE 2014. CCIS, vol. 477, pp. 190–201. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13036-1_17
47. Heeager, L.T., Nielsen, P.A.: Meshing agile and plan-driven development in safety-critical software: a case study. Empir. Softw. Eng. **25**(2), 1035–1062 (2020). https://doi.org/10.1007/s10664-020-09804-z
48. Badanahatti, A., Pillutla, S.: 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, pp. 1–5. Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3385032.3385047

49. Łukasiewicz, K., Górski, J.: Introducing agile practices into development processes of safety critical software. In: Proceedings of the 19th International Conference on Agile Software Development: Companion, pp. 1–8. ACM, Porto Portugal (2018). https://doi.org/10.1145/3234152.3234174
50. Fitzgerald, B., Stol, K.-J., O'Sullivan, R., O'Brien, D.: Scaling agile methods to regulated environments: an industry case study. In: 2013 35th International Conference on Software Engineering (ICSE), pp. 863–872. IEEE, San Francisco, CA, USA (2013). https://doi.org/10.1109/ICSE.2013.6606635
51. Ambler, S.W.: Agile Model Driven Development (AMDD) (2007)
52. 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
53. Bryant, S., Romero, P., du Boulay, B.: Pair programming and the mysterious role of the navigator. Int. J. Hum.-Comput. Stud. **66**, 519–529 (2008). https://doi.org/10.1016/j.ijhcs.2007.03.005
54. Hulkko, H., Abrahamsson, P.: A multiple case study on the impact of pair programming on product quality. In: Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005, pp. 495–504 (2005). https://doi.org/10.1109/ICSE.2005.1553595
55. Chong, J., Plummer, R., Leifer, L., Klemmer, S.R., Eris, O., Toye, G.: Pair programming: when and why it works. 6 (2005)
56. Vanhanen, J., Mäntylä, M.V.: A systematic mapping study of empirical studies on the use of pair programming in industry. Int. J. Softw. Eng. Knowl. Eng. **23**, 1221–1267 (2013). https://doi.org/10.1142/S0218194013500381
57. Sobral, S.R.: Is pair programing in higher education a good strategy? Int. J. Inf. Educ. Technol. **10**, 911–916 (2020). https://doi.org/10.18178/ijiet.2020.10.12.1478
58. Williams, L., Kessler, R.R., Cunningham, W., Jeffries, R.: Strengthening the case for pair programming. IEEE Softw. **17**, 19–25 (2000). https://doi.org/10.1109/52.854064
59. Williams, L.A., Kessler, R.R.: Experiments with industry's "Pair-Programming" model in the computer science classroom. Comput. Sci. Educ. **11**(1), 7–20 (2001)
60. Arisholm, E., Gallis, H., Dyba, T., Sjoberg, D.I.K.: Evaluating pair programming with respect to system complexity and programmer expertise. IEEE Trans. Softw. Eng. **33**, 65–86 (2007). https://doi.org/10.1109/TSE.2007.17
61. Nosek, J.T.: The case for collaborative programming. Commun. ACM. **41**, 105–108 (1998). https://doi.org/10.1145/272287.272333
62. Ciolkowski, M., Schlemmer, M.: Experiences with a Case Study on Pair Programming. 7 (2002)
63. Dongo, T.A., Reed, A.H., O'Hara, M.T.: Exploring pair programming benefits for MIS majors. J. Inf. Technol. Educ.: Innovations Pract. **15**, 223–239 (2016). https://doi.org/10.28945/3625
64. Vanhanen, J., Lassenius, C., Mantyla, M.V.: Issues and tactics when adopting pair programming: a longitudinal case study. In: International Conference on Software Engineering Advances (ICSEA 2007), p. 70. IEEE, Cap Esterel, France (2007). https://doi.org/10.1109/ICSEA.2007.48
65. Phaphoom, N., Sillitti, A., Succi, G.: Pair programming and software defects – an industrial case study. In: Sillitti, A., Hazzan, O., Bache, E., Albaladejo, X. (eds.) XP 2011. LNBIP, vol. 77, pp. 208–222. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20677-1_15
66. di Bella, E., Fronza, I., Phaphoom, N., Sillitti, A., Succi, G., Vlasenko, J.: Pair programming and software defects–a large, industrial case study. IEEE Trans. Softw. Eng. **39**, 24 (2013)
67. Jensen, R.: A pair programming experience. undefined (2003)

68. Vanhanen, J., Lassenius, C.: Perceived effects of pair programming in an industrial context. In: 33rd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO 2007), pp. 211–218. IEEE, Lubeck, Germany (2007). https://doi.org/10.1109/EUROMICRO.2007.47
69. Vanhanen, J., Korpi, H.: Experiences of using pair programming in an agile project. In: 2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07). pp. 274b–274b (2007). https://doi.org/10.1109/HICSS.2007.218
70. Phongpaibul, M., Boehm, B.: A replicate empirical comparison between pair development and software development with inspection. In: First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007), pp. 265–274 (2007). https://doi.org/10.1109/ESEM.2007.33
71. Fagan, M.E.: Advances in software inspections. IEEE Trans. Softw. Eng. **SE-12**(7), 744–751 (1986). https://doi.org/10.1109/TSE.1986.6312976
72. Vanhanen, J., Lassenius, C.: Effects of pair programming at the development team level: an experiment. In: 2005 International Symposium on Empirical Software Engineering, 2005. p. 10 (2005). https://doi.org/10.1109/ISESE.2005.1541842
73. Sison, R.: Investigating the effect of pair programming and software size on software quality and programmer productivity. In: 2009 16th Asia-Pacific Software Engineering Conference, pp. 187–193. IEEE, Batu Ferringhi, Penang, Malaysia (2009). https://doi.org/10.1109/APSEC.2009.71
74. Tomayko, J.E.: A comparison of pair programming to inspections for software defect reduction. Comput. Sci. Educ. **12**, 213–222 (2002). https://doi.org/10.1076/csed.12.3.213.8614
75. Balijepally, V., Mahapatra, R., Nerur, S., Price, K.H.: Are two heads better than one for software development? The productivity paradox of pair programming. MIS Q. **33**, 91 (2009). https://doi.org/10.2307/20650280
76. Sison, R.: Investigating pair programming in a software engineering course in an asian setting. In: 2008 15th Asia-Pacific Software Engineering Conference, pp. 325–331. IEEE, Beijing, China (2008). https://doi.org/10.1109/APSEC.2008.61
77. Padmanabhuni, V.V.K., Tadiparthi, H.P., Yanamadala, M., Madina, S.: Effective pair programming practice- an experimental study **3**, 9 (2012)
78. Phongpaibul, M., Boehm, B.: An empirical comparison between pair development and software inspection in Thailand. In: Proceedings of the 2006 ACM/IEEE international symposium on International symposium on empirical software engineering - ISESE '06, p. 85. ACM Press, Rio de Janeiro, Brazil (2006). https://doi.org/10.1145/1159733.1159749
79. Winkler, D., Kitzler, M., Steindl, C., Biffl, S.: Investigating the impact of experience and solo/pair programming on coding efficiency: results and experiences from coding contests. In: Baumeister, H., Weber, B. (eds.) Agile Processes in Software Engineering and Extreme Programming: 14th International Conference, XP 2013, Vienna, Austria, June 3–7, 2013. Proceedings, pp. 106–120. Springer Berlin Heidelberg, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38314-4_8
80. Madeyski, L.: Impact of pair programming on thoroughness and fault detection effectiveness of unit test suites. Softw. Process Improv. Pract. **13**, 281–295 (2008). https://doi.org/10.1002/spip.382
81. Arcos-Medina, G., Mauricio, D.: The influence of the application of agile practices in software quality based on ISO/IEC 25010 standard. Int. J. Inf. Technol. Syst. Approach. **13**, 27–53 (2020). https://doi.org/10.4018/IJITSA.2020070102
82. Pandit, P., Tahiliani, S.: AgileUAT: a framework for user acceptance testing based on user stories and acceptance criteria. Int. J. Comput. Appl. **120**, 16–21 (2015). https://doi.org/10.5120/21262-3533

83. Miller, R.W., Collins, C.T.: Acceptance Testing (2002)
84. Scott, E., Tõemets, T., Pfahl, D.: An empirical study of user story quality and its impact on open source project performance. In: Winkler, D., Biffl, S., Mendez, D., Wimmer, M., Bergsmann, J. (eds.) SWQD 2021. LNBIP, vol. 404, pp. 119–138. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-65854-0_10
85. Lucassen, G., Dalpiaz, F., van der Werf, J.M.E.M., Brinkkemper, S.: Improving agile requirements: the quality user story framework and tool. Requirements Eng. **21**(3), 383–403 (2016). https://doi.org/10.1007/s00766-016-0250-x
86. Jeeva Padmini, K.V., Perera, I., Dilum Bandara, H.M.N.: Applying agile practices to avoid chaos in user acceptance testing: a case study. In: 2016 Moratuwa Engineering Research Conference (MERCon), pp. 96–101. IEEE, Moratuwa, Sri Lanka (2016). https://doi.org/10.1109/MERCon.2016.7480122
87. Duka, D.: Agile Experiences in Software Development 6 (2012)

Author Queries

Chapter 5

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.	
AQ2	As per Springer style, both city and country names must be present in the affiliations. Accordingly, we have inserted the city name "Dundalk, Newtownabbey" in affiliation "1, 2" respectively. Please check and confirm if the inserted city name "Dundalk, Newtownabbey" are correct. If not, please provide us with the correct city name.	