The DIFFC Model: Addressing documentation challenges in Large-Scale Agile Requirements Engineering

Agnes Aba Wadee
(6398197)

Master's Thesis (Final version)

MSc. Business Informatics

Department of Information and Computing sciences,
Utrecht University, The Netherlands

Dr. Fabiano Dalpiaz (First supervisor)
Dr. Gerard Wagenaar (Second supervisor)

Utrecht University

September 2022

Abstract

Agile Requirements Engineering (RE) addresses several challenges in plan-driven RE but poses new challenges such as minimal documentation. With less focus on documentation and more focus on developing working software, documentation tends to be overlooked. In this research, we sought to address the challenges found in documentation by exploring the impact of Agile RE activities on the documentation in multi-team software projects. We conduct documentation analyses and eight semi-structured interviews with experts from two multi-team Agile Software Development (ASD) projects. The findings address the variations in how ASD is adopted at scale and the RE activities that are embedded in these practices as well as the documentation artefacts used. We identify the challenges and strengths facing the documentation artefacts and the documentation processes within the teams. Previous studies on software process initiatives address process improvement in agile teams but hardly address the issues found in the documentation. Therefore, we propose the DIFFC model, a lightweight treatment that focuses on the strengths of feedback within agile teams. Our model is validated via an experiment in two multi-team software projects. As a first step, the results are promising with potential for improvement in future research. Our treatment may assist practitioners in addressing the issues found in their documentation and improving their documentation processes.

Keywords: Agile Requirements Engineering, Requirements Engineering (RE), Agile Software Development (ASD), Large-scale Agile Software Development, Software Documentation, Software Process Improvement

Acknowledgements

It has been a long eight-month journey with highs and lows, but I could not have achieved a successful completion of this thesis without the help of many. To begin, I would like to express my gratitude to my first supervisor, Dr Fabiano Dalpiaz, for his frequent and detailed feedback and his help any time I had questions. There were moments when I had difficulties moving forward, but after our discussions, it became clear how to proceed. It was a pleasant atmosphere working together and I cannot thank him enough for all his help and support. Secondly, I would like to thank my daily supervisor for his feedback and the discussions that helped shaped the case studies. Also, many thanks to my second supervisor, Dr Gerard Wagenaar for all his feedback that helped shaped this research. Again, my thanks go to the members of the RE lab of Utrecht University for their fruitful discussions and tips that helped this research.

Combining a full-time study with a part-time job was not always easy. I could not have made it this far without the constant support and encouragement from my husband, Joel. His prayers, advice, and always seeing the best in me, were reasons to keep me going. Many thanks to my sisters, Adjoa and Johanna, my friends, Taofeeqat, Maartje, Jahmilla, and my family for being of great encouragement and support throughout this journey. Also, to my church family for all their prayers and encouragement throughout this journey.

Last but not least, this research would not have been a success without the enthusiasm and participation of the experts from the projects, who dedicated their time to the interviews and the experiment. I am forever grateful for their willingness to help. Lastly, I express my gratitude to my managers who gave valuable feedback on this thesis, and my colleagues for being flexible and coping with my thesis schedule.

Agnes Wadee, 9th September 2022.

Acronyms

ASD	Agile Software Development.
ASM	Agile Scaling Method.
AUP	Agile Unified Process.
BDD	Behavioural Driven Development.
CMM	Capability Maturity Model.
CoE	Centre of Excellence.
CoP	Communities of Practices.
CSD	Continuous Software Delivery.
DAD	Disciplined Agile Delivery.
DIFFC	Documentation Improvement Framework via Feedback Cycles.
DoR	Definition of Ready.
DSDM	Dynamic Software Development Method.
GQM	Goal-Question-Metric.
JIT	Just-in-time.
KIT	Key Informant Technique.
LeSS	Large-Scale Scrum.
LSOs	Larger Software Organisations.
LSPs	Larger Software Projects.
RE	Requirements Engineering.
S@S	Scrum@Scale.
SAFe	Scaled Agile Framework.
SDLC	Software Development Life Cycle.
SPD	Software Project Data.
SPI	Software Process Improvement.
TDD	Test Driven Development.
XP	eXtreme Programming.

Contents

Chapter 1

Introduction

In a digital age, where information is constantly changing at a rapid pace, documentation is essential to the success of software projects [1]. This thesis proposes a framework to address the issues found in documentation by exploring the impact of requirements engineering practices on documentation in large-scale software projects that adopt agile software development. Our research is positioned at the intersection of three domains: agile software development, requirements engineering, and software documentation.

The structure of this chapter is as follows. First, we introduce the problem statement in Section 1.1. Subsequently, we present the research questions in Section 1.2. Then, we highlight the expected contributions and an overview of the remainder of this thesis in Sections 1.3 and 1.4, respectively.

1.1 Problem Statement

Over the past two decades, there has been an increase in the number of companies adopting agile methods in software development projects [2]. The shift has been from traditional methods, such as waterfall, to agile methods such as Scrum, feature-driven development, eXtreme programming, among others [3]. Agile methods are guided by the principles in the Agile Manifesto. These principles distinguish agile from traditional approaches by improving customer collaboration, ensuring less focus on processes and more focus on the strengths and creativity of personnel [4]. Additionally, these principles ensure flexibility in coping with change and that developers focus more on having working software rather than extensive documentation to deliver features to the customer rapidly [5].

The success of an agile method is context specific. Agile methods are not sufficient on their own for large and complex environments, requiring them to be tailored and combined with other methods to address the complete software delivery cycle [6]. This is because agile methods were originally targeted at small, co-located, and self-organising teams who develop software in small iterations and in close collaboration with the customers [7]. Also, Kettunen (2007) argues

that "no one agile software method is in practice a complete solution for all situations" [7, p. 542].

Requirements engineering (RE) activities are practised in agile software development but they are informally used and the quality of use depends on the skills and expertise of the individuals [8]. In this research, we focus on the RE practices in large-scale (multi-team) agile software development. The reason why we scope our research to multi-team projects is that the larger the project, the greater the need is for formal documentation, which is contrary to the principles of agile software development [9]. Large software projects tend to have more dependencies between projects and teams compared to smaller organisations [9]. Hence, the scarcity of detailed and well-written documentation can be detrimental to large-scale software development [10]. Moreover, RE activities are considered one of the crucial activities in software development, because the problems enclosed in the system as a result of the elicited requirements are the most expensive to solve [11]. Hence, software engineers should strive to develop high-quality systems using techniques and tools [12]. Although agile RE practices address several challenges in traditional RE, they also pose new RE challenges including minimal documentation [13]. With less focus on documentation and more focus on informal communication and developing working software at the end of each iteration, insufficient documentation in agile methods is a limitation in multi-team software projects [10]. Important features that are not identified upfront in agile methods may be forgotten or misunderstood and they might require rework later on [10].

Although software documentation is one of the oldest practices in software engineering, it is often neglected [14]. The process of writing documentation is perceived as a burdensome intrusive side-task and it is usually submitted a posteriori [15]. Developers tend to focus on delivering working software for a deadline and there is hardly enough time to document the decisions and knowledge after the project has been delivered [16]. Besides, the Agile Manifesto emphasises the delivery of the working software over comprehensive documentation [5]. Even though Agile Software Development (ASD) methods are described as lightweight, there are many artefacts scattered throughout the entire ecosystem of tools as opposed to being documented in a single self-contained document [17]. These artefacts play a crucial role in the various phases of the software development life cycle [18]. They serve as a means of communication and knowledge transfer in software development [19]. Since documentation is overlooked in agile software development, it is even more important in large-scale agile methods as verbal communication is not as effective across teams in comparison with a smaller project. Hence, future research suggestions propose the need to optimise the documentation processes [20]. With that, we introduce our research questions in the next section.

1.2 Research Questions and Objectives

The objective of this research is to investigate the impact of requirements engineering practices on documentation in multi-team ASD projects. The research begins with a literature study, followed by a design science approach [21], adopting a multiple-case study [22] for problem investigation. The multiple-case study

collects and analyses data on two cases (software projects). The case studies provide insights into the variations in agile software development in multi-team software projects, the requirements engineering practices, and the current state of documentation in the projects. Design science treatment design and validation are then used to design and validate a framework to address documentation issues by improving the documentation practices in large-scale agile software development. The research approach is explained in more detail in Chapter 3.

The main research question is phrased as:
How can the challenges facing documentation due to Agile RE activities in multi-team software projects be addressed?
The main question is divided into seven sub-questions as presented in Table 1.1. These questions are used as a guide for this design science research.

Table 1.1: An overview of sub-research questions with the rationale.

Sub-research Question	**Rationale**
SRQ1: How is Agile Software Development (ASD) adopted within a multi-team software project?	Agile methods were targeted at small, co-located, and self-organising teams who develop software in small iterations and in close collaboration with the customers [7]. And "no one agile software method is in practice a complete solution for all situations" [7, p. 542].
SRQ2: Which requirements engineering practices can be identified in the adopted ASD approach?	RE is the most important phase of the software development life cycle [23]. However, there is a knowledge gap in the role of RE in agile methods [24].
SRQ3: Which kinds of documentation artefacts are used for the various requirements engineering practices within the ASD approach?	Although ASD methods are described as lightweight, there are many artefacts scattered throughout the entire ecosystem of tools as opposed to being documented in a single self-contained document [17]. Additionally, hybrid methods combine activities from traditional plan-driven methods with activities from agile methods [25], and this may impact the kind of documentation used in the project.
SRQ4: What is the current state of the identified documentation (from SRQ3) within the project?	Software documentation is one of the recommended and oldest practices in software engineering, yet it is often lacking [14]. We aim to analyse the quality of documentation artefacts used in agile software development.
SRQ5: What is the impact of requirements change on the identified documentation (from SRQ3) within the project?	How agile teams respond to requirements change in agile software development is still a research gap [26]. Even though the Agile Manifesto does not recommend extensive documentation, defining requirements change in detail is a necessary approach to ensure a better understanding of the requirements change [26].
SRQ6: What framework can be used to address documentation issues by improving documentation practices in multi-team software projects?	To the best of our knowledge, there is no Software Process Initiative that addresses the challenges facing documentation in agile teams. We design a framework to address the issues found in documentation in multi-team ASD projects.
SRQ7: What is the effectiveness of the proposed framework from SRQ6?	We validate the treatment by practitioners to ensure sound validity and solid contributions to the literature and industry.

1.3 Expected Contributions

We contribute to the theoretical body of agile software development by providing evidence of the variations of the adoption of ASD in multi-team software projects by means of a multiple-case study. Here, we identify the various activities, roles, and tools in the adoption of scaled-ASD in the explored cases. Additionally, we gather empirical evidence of requirements engineering activities in multi-team ASD projects with the accompanying artefacts and rationales for use. Furthermore, we tackle the knowledge gap in software documentation by providing evidence of the issues of documentation in multi-team software projects. We expand the theory on the documentation artefacts and rationales of use in ASD as well as the challenges facing documentation, by providing empirical data from the explored multi-team software projects. Finally, we enrich the literature with a proposed framework that can be used to address the issues found in documentation in multi-team ASD projects. This proposed framework is also applied in an industrial setting to validate it based on its effectiveness.

Our findings can also give practitioners insights into the common practices and differences of agile methods in multi-team software projects. More importantly, our findings will serve as recommendations for practitioners to improve their documentation practices in an attempt to approach the renowned documentation issues in the software industry.

1.4 Thesis Outline

The remainder of this thesis is structured as follows. We present the knowledge on previous studies relating to the software development life cycle, ASD methods, agile requirements engineering, scaled agile methods, and software documentation in Chapter 2. This research uses a design science qualitative approach adopting a multiple-case study for problem investigation on a study of two distinct cases. The research approach is discussed in Chapter 3. Then, we present our findings from the case study in Chapters 4 and 5. Where in Chapter 4, we introduce the teams and projects of each case along with the scaled ASD approach, RE activities and documentation used in those projects. And in Chapter 5, we elaborate on the current state, strengths and potential areas for improvement of the identified documentation within the project. With the insights from the Problem Investigation phase and a semi-systematic literature study, we design and propose a treatment with both scientific and industrial implications in Chapter 6. Subsequently, we present how we validate the proposed framework in Chapter 7. Lastly, our conclusions and discussions are discussed in Chapters 8 and 9.

Chapter 2

Related Literature

We review the main knowledge that our research is built upon. The research topic covers literature on Agile Software Development, Requirements Engineering, and Software Documentation. We apply a semi-systematic literature review approach to select primary and secondary studies for this research on the aforementioned topics. A semi-systematic approach is adopted for "topics that have been differently conceptualised and studied by a different group of researchers" [27, p. 988]. Those topics are ones which have been studied with different approaches and perspectives in the literature [27]. The study began by defining the literature review research questions. Then, fairly recent systematic literature review papers such as [17], [24], [28], and [29] were studied. These papers gave insights into primary studies that are relevant to this research. Searches were conducted using Scopus and Google Scholar. The backwards and forward references of the selected papers were also traced to seek relevant sources for this research.

The literature review tackles the following questions.

- How do software development methods evolve from the Software Development Life Cycle (SDLC)?
- What are the characteristics of agile methods?
- To what extent is requirements engineering evident in agile methods?
- How is agile software development implemented in multi-team software projects?
- What are the benefits and challenges facing software documentation?
- Which documentation artefacts and practices can be found in agile software development?
- What kind of tools are used for documentation in agile software development?

The SDLC is discussed as well as various early development methods in Section 2.1. Subsequently, agile methods such as eXtreme Programming, Scrum, and Kanban are introduced in Section 2.2. The evidence of Requirements Engineering practices in Agile Software Development is elaborated upon in Section 2.3. From the literature, we see that agile methods were originally made to suit smaller and co-located project teams. Hence, we discuss agile methods in large software projects in Section 2.4. Finally, related literature on the software documentation aspects of the research topic such as the benefits, challenges, artefacts, and tools are discussed in Section 2.5.

2.1 The Software Development Life Cycle

The Software Development Life Cycle (SDLC) is a method that organises the development of software into different phases in a systematic manner to increase the probability of completing the software project within a specific time while maintaining the quality of the software product [30]. The terms software development life cycle and system development life cycle are intertwined in the literature. The abbreviation "SDLC" is used to describe either of those depending on the context [31]. Although the concepts are similar according to Ruparelia (2010), the main difference is that the first describes the life cycles of software, whereas the latter refers to the life cycle of a system that includes software development [31]. The system development life cycle consists of four main phases and is used during the development of any system [32]. These phases are identified as analysis, design, implementation, and testing [32]. In this research, we use SDLC to refer to the Software Development Life Cycle. The choice of a suitable SDLC model depends on factors such as organisational policies, stakeholders' concerns, and stakeholder preferences, among others. An SDLC model depicts a set of SDLC activities performed in an ordered manner to produce the desired software product [30]. Each SDLC model has its strengths and weaknesses and may be more suitable in one situation than in another [30]. In general, each SDLC model consists of the requirements gathering, design, implementation, testing, deployment, and maintenance phases [30].

However, with the increasing need for continuous and rapid deployment of software, there is an increasing shift from the application of the traditional SDLC model where the testing, deployment, and maintenance phases are separated [33]. In the past decade, DevOps is being adopted more and more by the industry. 'DevOps is a set of methods in which developers and operations communicate and collaborate to deliver software and services rapidly, reliably and with higher quality' [34, p. 1]. It is a blend of Development and Operational activities which seeks to integrate both activities to empower software development teams 'with full accountability of their service and its underlying technology stack; from development, to deployment and support' [34, p. 1].

2.1.1 Evolution of Software Development Methods

The Waterfall Model is the oldest and most well-known method depicting the SDLC [35]. It was introduced in 1970 by Royce and is distinguished by its linear and sequential structure of phases [30]. A phase of this model begins and ends before starting the next and the phases do not overlap [35]. In Figure 2.1, the Waterfall Model is shown, the phases are identified as Requirements Analysis, Design, Implementation, Testing, and Deployment & Maintenance [30]. Waterfall is selected for use in projects when the requirements are well known, very clear, and the quality of the project is more important than cost or time [35]. There is extensive documentation upfront when using this SDLC model, to ensure requirements and constraints are well documented. Previous studies describe Waterfall as heavyweight, highlighting that the bureaucratic nature of this model is one of its greatest downfalls [36]. Also, all the requirements need to be rigidly defined in the beginning. Additionally, the project manager is expected to foresee and document all the potential risks and problems upfront

which is not practical [36]. Thereby, unforeseen events can negatively impact the time, budget, and quality concerns of the project [36]. In response to the weaknesses and failures of Waterfall, many new methods have emerged to add some iteration to the software development process.

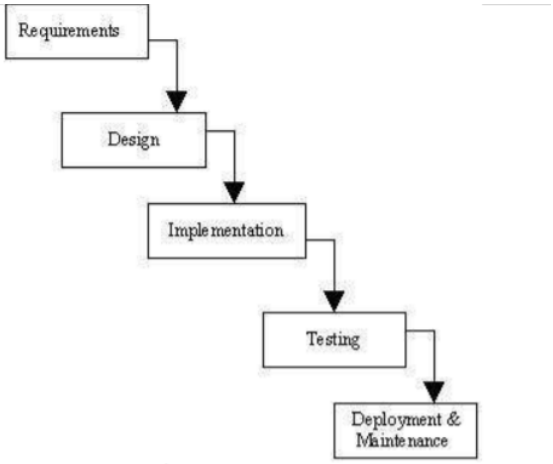


Figure 2.1: The Waterfall Model adopted from [30].

An example of a response to the sequential flow of waterfall is the iterative model. Here, the elements of the waterfall method are combined in an iterative manner [35]. The requirements are not completely elicited and the iterative process begins with a small set of those requirements [37]. Each iteration develops a small part of the product until the final version is developed [37]. Figure 2.2 presents an overview of the iterative model. Iterative and incremental development is used when the entire requirements of the system are not clear, and when essential requirements must be implemented but some functionality is subjected to change with time [37].

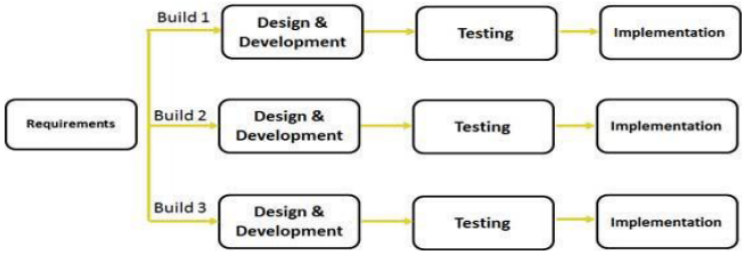


Figure 2.2: The iterative model adopted from [37].

The systematic combination of iterative and waterfall development forms the spiral process model [37]. Figure 2.3 illustrates an overview of the spiral process model. Spiral development consists of four phases which are planning, risk analysis, development, and evaluation [35]. The *identification phase* includes the understanding of the system requirements which involves continuous communications between the customers and the system analysts [35]. In the *risk*

analysis phase, the risks in the project are assessed along with potential solutions and a prototype is delivered at the end of this phase [35]. Subsequently, the software is developed and tested in the *development phase* [35]. The outcome of the development phase is evaluated by the customer before the project continues to the next spiral [35].

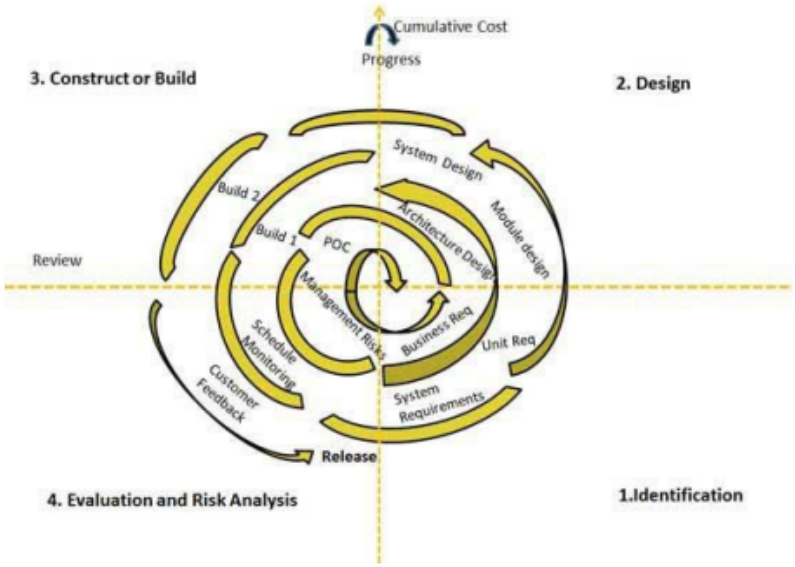


Figure 2.3: The spiral process model adopted from [37].

Nonetheless, the shift has been from traditional methods, such as waterfall, to incremental software development methods such as Scrum, feature-driven development, eXtreme Programming (XP), Dynamic Software Development Method (DSDM), Test Driven Development (TDD), Crystal method, Adaptive Software Development, and Agile Unified Process (AUP) [3], [4], [36], [38]. These incremental models are also referred to as agile development methodologies or lightweight methodologies [38]. In the research by Abrahamsson et al. (2003), the authors present the life-cycle and evolution of agile methods which can be found in Figure 2.4. In this figure, we see the transition from iterative approaches such as the Spiral Model to adaptive methods such as Adaptive Software Development and to Agile Software Development, which gave rise to Agile Modeling.

2.2 Agile Development Methods

Agile Software Development (ASD) is an adaptive, iterative, and incremental approach to software development [4]. It is a group of software development methods based on a set of best practices as stated as principles in the "Agile Manifesto". These principles distinguish ASD from traditional SDLC models by improving customer collaboration, ensuring less focus on processes and more focus on the strengths and creativity of people [4]. Also, these principles ensure flexibility in coping with change and that the developers focus more on having working software rather than extensive documentation [5]. In Table 2.1, we

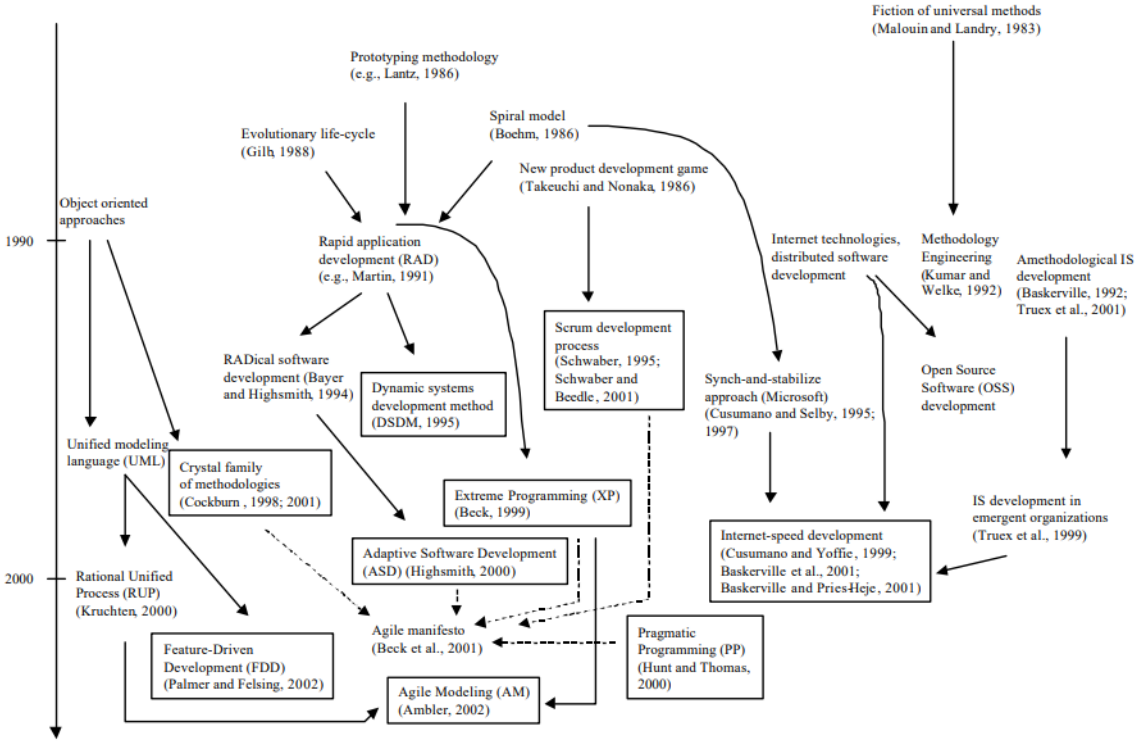


Figure 2.4: An overview of the evolution of Agile methods. Retrieved from [39, p. 246].

present the values and their descriptions from the Agile Manifesto.

As discussed in Section 2.1, agile software development comprises various methods. According to a survey presented in the 15th State of Agile Report on the usage of Agile methods, Scrum is the most popular ASD method with 66% followed by ScrumBan, Kanban, and Scrum/XP Hybrid[1]. Figure 2.5 depicts the results of their survey. Hence, these results confirm the popularity of Scrum among the other Agile methods in previous studies. In the subsequent subchapters, we discuss the characteristics of the main methods; Scrum, Kanban, and XP.

[1]The 15th State of Agile Report presents current trends in agile software delivery in 2021. Retrieved from https://itnove.com/wp-content/uploads/2021/07/15th-state-of-agile-report.pdf on 18th January 2022.

Table 2.1: Values in ASD adopted from [5]

Value	Description
Customer collaboration over contract negotiation	Reduce formalities to start and finish faster, with a strong focus on the customer throughout the development process.
Individuals and interactions over processes and tools	Enhance communication within the teams and remove barriers.
Working software over comprehensive documentation	Developers should spend more time implementing and testing activities rather than writing extensive documentation.
Responding to change over following a plan	Give teams the freedom to make changes and adjust to project needs.

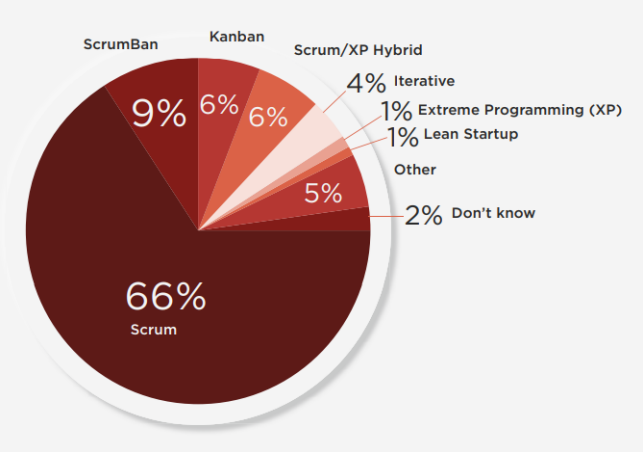


Figure 2.5: The state of the 15th Agile Survey Results on the use of Agile methods. Adopted from[1].

2.2.1 EXtreme Programming (XP)

XP is a lightweight ASD method suitable for small teams with the capability to respond quickly to changing requirements [40]. It was introduced in the late 1990s by Kent Beck, the pioneer of the Agile Manifesto. The main activities of XP are shown in Figure 2.6. Developers use pair-programming to code the functionalities of the software product and the focus is only on implementing the functional requirements [40]. The pair-programming activity leads to collective code ownership. This implies that anyone can modify the code at any time since the ownership of the code is shared [40]. Hence, the 'courage' principle of XP infers the correction and removal of errors from the code at all costs [41]. The other principles of XP discussed by [41] are:

- Communication: A core characteristic of communication is pair-programming as mentioned earlier. Also, a role is assigned as an XP coach to detect communication failures and facilitate adequate communication [41]. Since communication is a very essential activity in this method.
- Simplicity: XP aims to develop software as easily and quickly as possible, thereby not dealing with functionalities that might be necessary in the future

are crucial at the moment [41].

- Feedback: To ensure feedback, testing is performed at all development stages rather than after the implementation phase [41]. Automated tests are used to ensure conformance with the given requirements.
- Respect: This principle closely relates to the emphasis on communication within XP. The team members should have an interest in the work of their colleagues and collaborate positively [41].

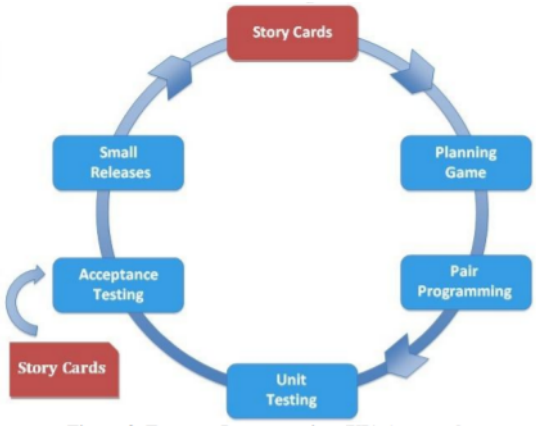


Figure 2.6: An overview of the activities in XP. Adopted from [42].

2.2.2 Scrum

Scrum structures the software development activities in short iterations referred to as sprints [43]. An overview of the main activities of Scrum is illustrated in Figure 2.7. Each sprint potentially ends with an executable version of the software application which is used as an input for the next sprint [43]. Generally, each sprint includes many phases of the SLDC such as designing, implementation, testing, and customer verification [42].

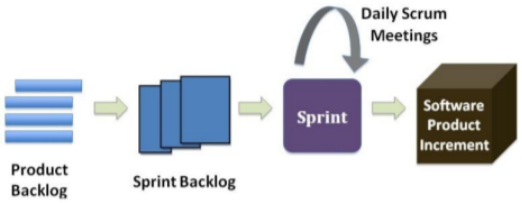


Figure 2.7: An overview of the activities in Scrum. Adopted from [42].

In line with the second Agile value, Scrum emphasises team collaboration. The development team is a cross-functional team consisting of people with different skills and expertise working together [43]. 'They are also self-organising, meaning they internally decide who does what, when, and how' [43, p. 5]. Additionally, a distinctive feature of Scrum is the short-duration daily meeting known as the Daily Stand-up or the Daily Scrum. Here, the members of the

development team discuss the status of their tasks [43] as well as any hindrances known as impediments. The requirements of the software product are elicited and stored in the product backlog [43]. This serves as a guide to plan the sprints. A subset of the requirements on the product backlog is used for the sprint backlog to define what needs to be implemented in that sprint [43].

2.2.3 Kanban

Kanban in software development was pioneered in 2004 by David Anderson while assisting a software development team at Microsoft [44]. Kanban is based on the just-in-time (JIT) philosophy [45]. The main idea of JIT is to continuously optimise the processes by ensuring waste reduction. Kanban focuses on visualising and limiting the work-in-progress in software development [44]. The main principles of Kanban are discussed as follows. Visualising work-in-progress ensures that the requirements are optimised in the development workflow by distinguishing between work to-do, in-progress and completed [42]. The visualisation of tasks is made possible by means of a tool known as the Kanban Board [42]. Furthermore, Kanban is complementary to Continuous Software Delivery (CSD) as it delivers software in increments rather than releasing software in batches [42]. Another unique characteristic of Kanban is the minimisation of waste. Tasks on the Kanban Board are only executed when they are required [42]. This results in a constant delivery of work items to the customers as the developers only focus on the necessary requirements at a given time [46].

Ahmad et al. (2015) investigated why two experienced Scrum teams shifted to Kanban by conducting seventeen semi-structured interviews with different teams at two large Finnish software companies [47]. In contrast to the survey results from the 15th Agile Survey, they found that companies are switching from Scrum to Kanban since it claims to offer improved visibility and quality of the software product as well as improved team motivation, communication and collaboration [47].

2.2.4 Feedback in ASD

As self-organising teams, a big part of the adaptability of agile teams to change is working in iterations and constantly improving based on feedback among the team members [48]. In a research by Matthies (2019) on feedback and process improvement approaches in ASD, feedback cycles are presented as a fundamental aspect of ASD methods as a driving factor for process improvement [48]. In ASD and specifically in Scrum, while there are scheduled meetings for discussions and feedback, there are also informal moments for feedback [48]. Scrum activities include meetings to facilitate the collection of feedback on the processes within the team but also on the outcome of the iteration [48]. Matthies (2019) presented an overview of the various feedback moments in Scrum which is shown in Figure 2.8. Here, it is illustrated that feedback is received at various cycles and time-frames ranging from seconds to months.

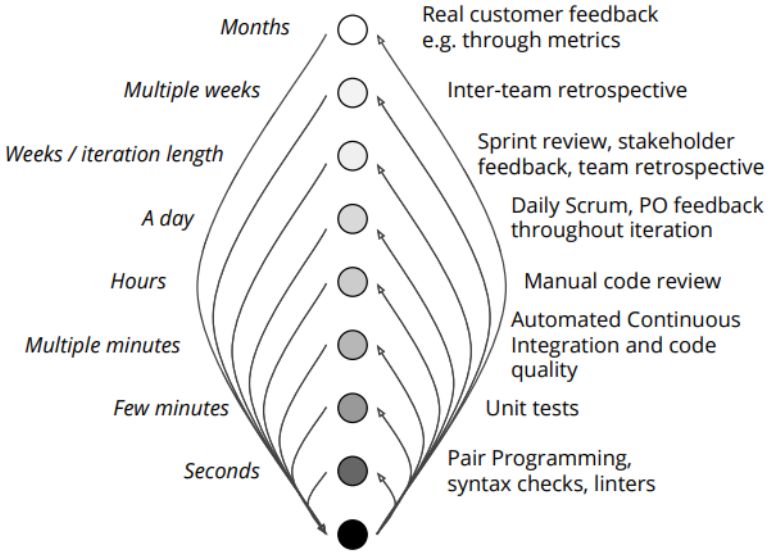


Figure 2.8: An overview of the various feedback cycles within Scrum. Retrieved from [48, p. 150].

2.3 Agile Requirements Engineering

Requirements engineering (RE) practices are evident in ASD but they are informally used and the quality of use depends on the skills and expertise of the individuals [8]. Agile requirements engineering is a term that defines the "agile way" of planning, executing and reasoning about requirements engineering activities [24]. RE has many definitions across literature, it can be defined as the process of identifying stakeholders and their needs, documenting them in a form to facilitate analysis, communication, and subsequent implementation [49]. There are multiple perspectives of the phases of RE in the literature. For this research, we adopt the phases discussed by Dietse (2009). RE entails the elicitation, analysis, specification, validation and management of software requirements [50]. We explain these terms briefly.

- *Elicitation* refers to identifying the problems that need to be solved [49]. Here, the stakeholders are identified as well as their objectives towards the to-be software system. Various techniques such as introspection, interviews, questionnaires, brainstorming, and prototyping are used to elicit requirements [51].
- *Analysis* deals with the activities such as conceptualising models or prototyping to ensure completeness of the requirements and capture an understanding of the organisation in question [52]. The understanding of an organisation entails its business rules, goals, tasks, and the necessary data [52].
- *Specification* is defined as an integral description of the behaviour of the to-be system. Scenarios, templates, use case modelling and natural language text are some of the most used techniques for requirements specification [53].
- *Validation* serves as an activity to ensure that the elicited requirements are an

accurate representation of the actual stakeholders' requirements [52]. Examples of techniques used to validate requirements are reviews and traceability [52].

- *Requirements management* involves recognising and managing the changes in requirements by a means of continuous requirements elicitation [54].

2.3.1 Agile RE Practices

In this sub-section, we identify the RE practices that are evident in ASD. In a systematic literature study by Inayat et al. (2015), literature published between 2002 and June 2013 was analysed to gather evidence on the RE practices adopted by agile teams as well as the challenges faced. It was observed that 17 RE practices were adopted in ASD [24]. These RE practices are shown in Table 2.2, along with a description of each practice. Figure 2.9 is adopted from the research of Ramesh et al. (2010), as a summary of the agile RE practices discussed in Table 2.2 and the challenges subsequently discussed in Section 2.3.2.

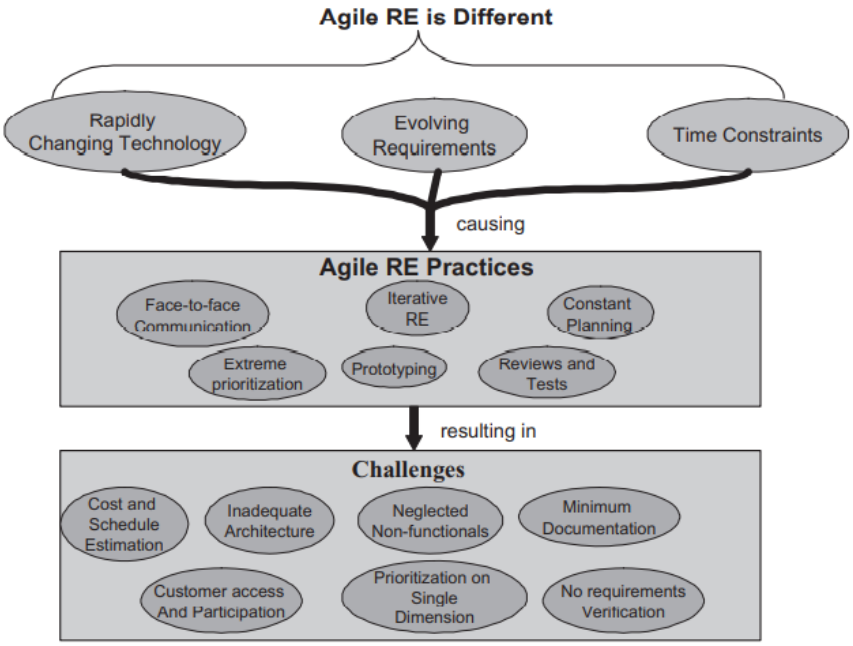


Figure 2.9: A summary of the agile RE practices and challenges. Retrieved from [55].

Table 2.2: RE practices in ASD adapted from [24]

RE practice	Description
Face-to-face communication	This type of communication is the main form of communication in agile methods advocated by the Agile Manifesto [56].
Customer involvement	Agile principles ensure strong customer collaboration throughout the development process rather than rigid contracts [5]. This practice hugely contributes to the success of the project [57].

Table 2.2: (continued from previous page)

RE practice	Description
User stories	Requirements in agile methods such as XP and Scrum are specified using user stories as artefacts [58]. These media facilitate communication and better overall understanding among stakeholders [59].
Iterative requirements	Requirements emerge over time in agile methods [55]. "Frequent interaction among stakeholders leads to this iterative requirements approach" [24, p. 922].
Requirement prioritisation	In contrast to traditional RE where requirements are prioritised before the start of development, requirements are continuously prioritised in each development cycle in agile methods [56].
Change management	"Change management has proven to be a significant challenge for traditional approaches thus far." [24, p. 922]. Agile methods facilitate the change of requirements, which is mainly the addition or removal of features [56].
Working in cross-functional teams	Agile teams comprise different expertise working together such as developers, testers, designers, and managers [24]. This practice ensures effective communication and helps reduce challenges such as over-scoping requirements [13].
Prototyping	Prototyping is a technique used for eliciting requirements in traditional RE. It promotes quicker feedback and enhances the customer anticipation of the software product [24].
Testing before coding	TDD is an approach whereby developers create tests before writing the functional code. It serves as a requirements activity due to the tests specifying the code's behaviour [56].
Requirements modelling	This practice is also performed in traditional RE, but executed with different techniques. In agile methods, techniques such as goal sketching [60], RE-KOMBINE [61], and user-story-based modelling [58] are examples of modelling techniques. They ensure quality and empower decision-making during the process of requirements negotiation [24].
Requirements management	In agile methods, requirements management is used to maintain the features on the product backlog for the instance of Scrum [24].
Review meetings and acceptance tests	Review meetings are a way of checking the status of completion of user stories in agile methods [24]. Additionally, acceptance tests are automated tests that result in binary results of success or failure for a user story [24].
Code refactoring	Code refactoring is used as a requirements engineering practice in agile methods as it accommodates the effect of changes to requirements on the code [62].
Shared conceptualisations	This RE practice supports the concept of carrying out RE activities related to elicitation, clarification, and change management [63]. Communication and collaboration are the basis of these concepts [24].
Pairing for requirements analysis	This practice is one of the ways to reduce close communication gaps between agile teams [24]. Here, stakeholders are encouraged to perform many roles during requirements analysis to ensure efficient task delegation and minimal communication delay [24], [64].
Retrospectives	Retrospectives are held at the end of each iteration [58]. The purpose of these meetings is to review the progress of the planned work as well as define future steps for new requirements as well as any rework [24].
Continuous planning	This practice ensures that the agile team is flexible to changes in requirements [24].

2.3.2 Challenges of Agile RE Practices

Although agile RE practices address several challenges such as communication gaps, over-scoping, and unreliable requirements specifications in traditional RE, they also pose new challenges to RE [13]. These challenges are discussed as follows.

The first challenge is minimal documentation. Software engineering methods suffer from the lack of sufficient documentation [14], and agile methods are no exception [56]. In ASD, emphasis is made on verbal communication rather than written specifications [5]. Instead of formal specifications, practitioners adopt the use of user stories in writing high-level requirements [58]. In Section 2.5.3, it is evident that there are many artefacts used in ASD. These artefacts are adopted for various rationales and are not only limited to the main artefacts of the agile method in use [25]. However, the use of formal documentation is not a barrier to frequent communication [56]. Rather, it steers the documentation of functional and technical artefacts such as functional designs, mock-ups, and technical designs [25]. In a study conducted by Stettina and Heijstek (2011), it was observed that practitioners found documentation very important, and disagreed with the principle of face-to-face communication over written specifications [1]. Minimal documentation can result in a lack of traceability and hinder knowledge transfer [65]. The challenges regarding documentation are discussed in detail in Section 2.5.2. Additionally, effective verbal communication between the agile team and the customer depends on factors such as "customer availability, the consensus among customer groups, and trust between the customer and the developers" [56, p. 63]. Besides, it was reported as a challenge to have the customer present on-site and most of the time product managers performed the role of surrogate customers [56]. Moreover, achieving consensus with different groups of stakeholders and customers is challenging and requires more effort to negotiate to achieve agreement with each group [56].

Another major challenge encountered in ASD is inappropriate architecture [24]. As requirements are introduced or modified, the initial architecture may become inadequate [55]. Hence, adaptation to the code may result in rework which can negatively impact the project cost [55]. Refactoring is a technique to modify the structure of the software in order to accommodate the new changes without affecting the observable behaviour of the software [56]. Although agile methods adopt refactoring as an RE practice, the need for refactoring is not always apparent [55]. It certainly introduces additional costs and does not completely address all the architectural issues [55].

Budget and time estimation is another challenge faced in agile methods. Since all the requirements are not known upfront in the beginning, the initial estimation is based on the known user stories. These requirements are subject to change during the course of the project. Hence, the original estimates require adjustment and that becomes a challenge for the management of the project [56]. Also, it can become a challenge for the customer to agree with the new estimations.

Subsequently, non-functional or quality requirements are often neglected when documenting requirements in agile methods [24]. They are often defined poorly and easily ignored during the initial development phases [55]. Although non-functional requirements (NFRs) define a variety of quality criteria, customers frequently focus on the functional requirements and ignore issues relating to NFRs. Even when NFRs are used, they tend to be documented and not tested. For instance, a requirement mentions the system should be available, without stating the criteria to test upon [55]. Ignoring quality requirements in the early phases of the project is likely to cause major issues as the software system

develops [55]. A common exception of the neglected NFRs is the usability of the system. Thanks to the high customer involvement in agile methods, there tends to be more feedback on the ease of use of the system [55].

Although requirements prioritisation is a practice adopted in ASD, it poses a challenge to agile methods. According to Ramesh et al. (2010), prioritisation is often based on the business value ensuring that the software is aligned with the business needs [55]. However, using only one dimension for prioritising requirements may result in issues relating to the scalability of the architecture, security and efficiency of the system. In the initial phases of the project, these requirements did not appear critical in comparison based on the business value, however, they became critical in the latter phases of the project [55].

Lastly, requirements validation in agile methods is a challenge. Although there is a high customer collaboration and involvement in requirements validation, aspects of verification such as the formalisation of detailed requirements are missing [55]. In general, agile methods do not prescribe artefacts for requirements validation and verification [55]. Hence, it becomes dependent on the agile team to use formal models of requirements for thorough verification [55].

2.4 Agile Methods in Large Software Organisations

The use of agile methods in software development has become continuously popular since the early 2000s [2]. With the increase in the number of successful projects in small software organisations and projects as a result of adopting agile methods, larger software organisations (LSOs) and larger software projects (LSPs) are increasingly adopting these methods [66]. As much as these methods address a significant amount of issues in traditional software methods, there are some hindrances in applying pure agile methods to large and/or software projects [10]. Even though a significant number of organisations have adopted agile methods, these methods have been criticised in the literature as being more suitable and applicable to smaller teams rather than LSOs [67]. Agile methods such as Scrum and XP are not sufficient on their own for large and complex environments, requiring them to be tailored and combined with other methods to address the complete software delivery cycle [6]. These methods were targeted at small, co-located, and self-organising teams who develop software in small iterations and in close collaboration with the customers [7]. The level of difficulty in adopting agile methods increases with organisational size [68]. Kettunen (2007) argues that "no one agile software method is in practice a complete solution for all (software project) situations" [7, p. 542]. Therein, the issues concerning the adoption of agile methods in LSOs and LSPs relate to the communication and scalability factors in these large software projects.

With less focus on documentation and more informal communication and developing a working software at the end of each iteration, insufficient documentation in agile methods is a limitation in LSPs. Important features that are not identified upfront in agile methods may be forgotten or misunderstood and it might require rework later on [10]. An example is when the architecture is inadequate to support the change, as discussed in Section 2.3.2. Moreover, it is difficult to

accurately estimate the time and resources needed for the project in the absence of a detailed plan [10]. For a smaller project, this risk is mitigated by planning in short iterations and close collaboration with the customer [10]. However, in LSPs, the magnitude of this risk is much higher [10], as it may significantly impact the cost and timeline of the project. In general, smaller projects require less formal updates and maintenance compared to LSPs. Hence, the scarcity of detailed documentation can be detrimental to software development on a larger scale [10]. LSOs tend to have more dependencies between projects and teams compared to smaller organisations [9]. The large size of the project also increases the need for formal documentation, which is contrary to the principles of agile methods [9].

The context of scalability in LSPs relates to the scalability of software engineering methods. The term is defined by Laitinen et al. (2009), as the "property of reducing or increasing the scope of methods, processes, and management according to the problem size" [69, p. 106]. The notion is that a software technique should be able to be scaled to suit a particular need, contractual requirement, and budgetary and business objectives [69]. It is no surprise that organisations are increasingly adopting the scaling of agile methods beyond single teams and projects [2]. There are a number of factors that contributes to the scalability of software projects which includes large multi-site, multi-customer, and multi-project organisations [70]. According to Leffingwell (2007), the challenges of scaling agile methods are distinguished into two categories, inherent to the method itself or external challenges imposed by the organisation [70]. The inherent challenges related to the fixed rules and assumptions of the method itself include issues such as the size and number of the teams, the availability of customers during the project, collocation, and architectural and documentation issues [70]. The external category of scalability issues relates to the challenges imposed by the organisation such as existing formalised policies and procedures, the coinciding of the agile method with existing process and project management organisations, corporate culture, as well as the fixed schedule and fixed functionality ordinance imposed by the organisation [70]. The term 'organisation' in this context refers to both the internal organisation and external stakeholders influencing the project.

In the literature, a number of methods have been proposed to address the issue of scalability in agile methods. The most commonly adopted method is the Scaled Agile Framework (SAFe) [2]. Other methods include the Disciplined Agile Delivery (DAD) [6], Agile Scaling Method (ASM) [6], Large-Scale Scrum (LeSS) [71], Scrum@Scale (S@S) [72], and Nexus [72] have been researched in previous studies. Also, evidence of internally created methods for scaling was found in the empirical study of Abheeshta Putta et al. (2021) on the adoption of agile scaling frameworks in LSOs [73]. Hybrid approaches are also evident whereby an agile method is combined with aspects from traditional software methods [25]. Therefore, the artefacts used in LSPs are a superset of those used in smaller projects. Examples of these artefacts are discussed in Section 2.5.3. Furthermore, we discuss a few of these agile scaling methods in detail as well as a comparison among them on principles and practices in the subsequent sub-sections.

2.4.1 Scaled Agile Framework (SAFe)

SAFe is known as the most common framework among the other agile scaling methods [2]. SAFe was introduced by Dean Leffingwell (2007) to scale agile methods to suit large enterprises. It offers four levels which are: *team*, *program*, *portfolio*, and *large solution* level [72]. At the team level, SAFe incorporates practices from agile methods such as Scrum, XP, and Kanban, but also from Lean. The agile teams are empowered to be self-organising and self-managing [74]. They work from a local product backlog, which is set up by the product owner(s) [74]. In Figure 2.10, we present an illustration of the SAFe model.

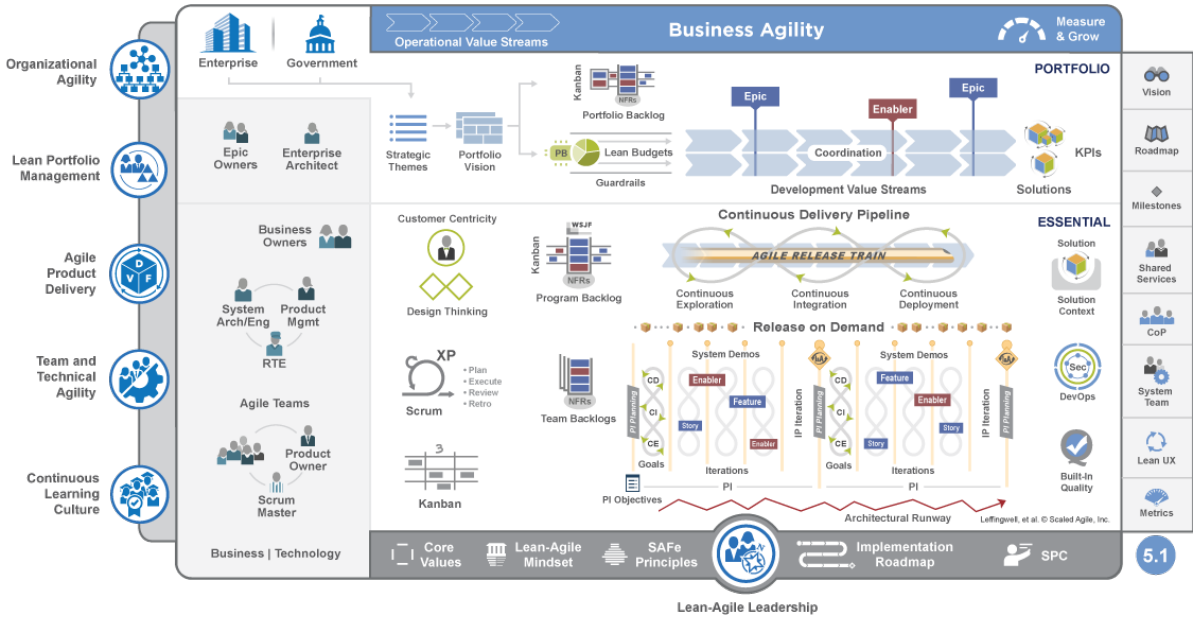


Figure 2.10: An overview of the activities of SAFe 5.1. Retrieved from *https://www.scaledagileframework.com/* on 6th September 2022.

At the program level, an additional set of challenges to execute agile at scale are addressed. These challenges are discussed by Leffingwell (2010), as maintaining the vision and roadmap, release management, quality management, deployment, resource management, and eliminating impediments [74]. Maintaining the vision and roadmap involves communicating the vision for the program and maintaining a roadmap to ensure the teams are aligned to a common goal [74]. Release management involves the organisation of the teams to build release increments on the enterprise's chosen development timeline [74]. Here, approximately five to twelve agile teams form an Agile Release Train to develop product increments, for the duration of eight to twelve weeks usually [72]. Also, resource management involves the adjustment of resources when necessary to address constraints in the program's ability to deliver the required value on time [74]. One of the questions that arise is "how to organise agile teams in order to optimise the agile teams in order to optimise value delivery of requirements [74, p. 64]. Leffingwell (2010) proposes the feature and component approaches to or-

ganising agile teams to address this concern [74]. A component team is usually responsible for a single layer in the architectural stack such as the presentation layer, business logic, or database [74]. On the contrary, the feature team works across the architectural stack to build features of the final product [74].

The portfolio level defines the program for the program level. Here, two new artefacts are introduced as investment themes and epics [74]. "Investment themes represent key product or service value propositions that provide marketplace differentiation and competitive advantage" [74, p. 84]. Epics are the high-level requirements artefacts that are used to coordinate development activities [74]. An investment theme defines epics, an epic defines features and a feature defines user stories [74]. These user stories are used by the agile teams as requirements [74]. Not only are these epics providing business value, but there are also epics concerning the architecture of the software in order to ensure a system that is robust to the change [74]. Lastly, the large solution level is an alternative or additional level to the portfolio level [72]. This level facilities the development of an integrated solution by a subset of the Agile Release Trains. At the large solution level, each epic needs a sponsor to fund the development whereas, at the portfolio level, solutions are grouped into value streams to realise business objectives [72]. SAFe provides a scalable requirements model for coordinating system behaviours among epics, capabilities, features, stories and non-functional requirements [2]. An overview of this model is shown in Figure 2.11.

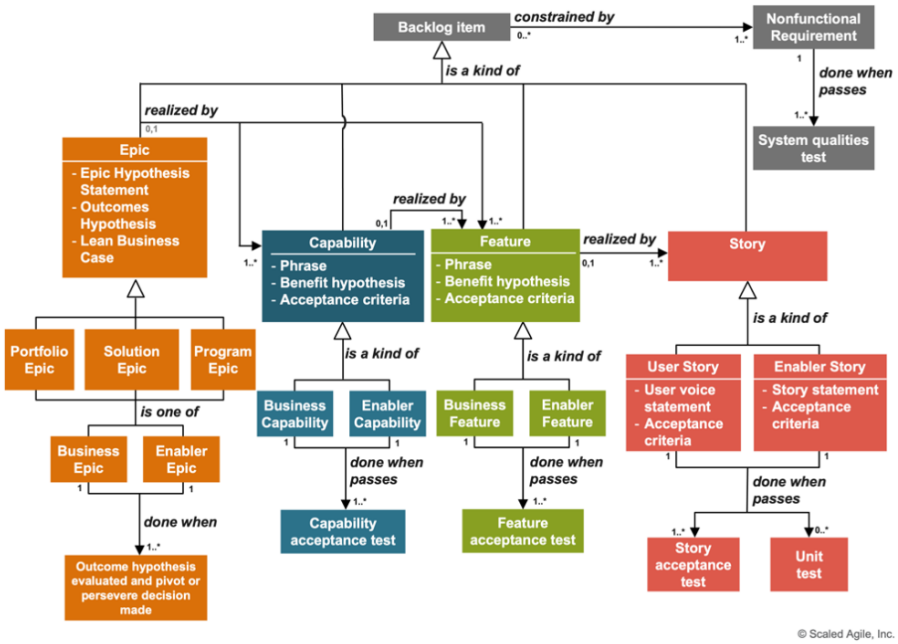


Figure 2.11: SAFe requirements model. Retrieved from *https://www.scaledagileframework.com/safe-requirements-model* on 28th August 2022.

[2] https://www.scaledagileframework.com/safe-requirements-model/

2.4.2 Disciplined Agile Delivery (DAD)

DAD is "an evolutionary (iterative and incremental) approach that regularly produces high-quality solutions in a cost-effective and timely manner via a risk and value-driven life cycle" [6, p. 14]. This approach distinguishes itself from other agile methods in four categories such as *a full delivery cycle, delivery of solutions rather than only software, a risk and value-driven life cycle, and self-organisation with an appropriate governance framework* [6]. DAD focuses on a full delivery life cycle of the project. The full delivery life cycle consists of smaller iterations. The life cycle of agile methods such as the Scrum life cycle focuses is on organising the work during a sprint [6]. However, this is not sufficient to meet the needs of the software delivery teams, but it is a foundation to develop a full delivery life cycle [6]. DAD builds upon the iterative approach to agile methods by recognising that agile delivery is *iterative in the small and serial in the large* [6]. Iterative in the small focuses on the daily iterative rhythm of work activities such as developing, testing, and modelling [6]. Serial in the large entails the structuring of the release rhythm through various phases such as initiation, development and deployment [6].

The second category that distinguishes DAD from agile methods is the delivery of solutions and not only software [6]. Teams using DAD deliver a complete solution which may include software, hardware, documentation, and/or the manual processes involved in working with the delivered system [6]. Moreover, the third category describes a risk and value-driven life cycle. In agile methods, the focus is on delivering working software at the end of each iteration [5]. However, DAD expands on this principle by identifying and actively mitigating the risks earlier in the life cycle [6]. For instance, the stakeholders agree on the scope of the project and the architecture of the system is designed and tested by building a working skeleton of the system [6]. Lastly, self-organisation with an appropriate governance framework refers to the activities involved in steering the activities and processes of the DAD team. The Agile Manifesto describes that the best architecture, requirements, and designs emerge from self-organising teams [5]. In parallel with self-organisation, DAD ensures appropriate governance that reflects the needs of the overall organisation such as having a common infrastructure and working towards organisational goals [6].

2.4.3 Agile Scaling Method (ASM)

ASM is introduced by Ambler (2009), as "a contextual framework for effective adoption and tailoring of agile practices for a system delivery team of any size" [6, p. 9]. It defines a guide to effectively adopt and tailor agile strategies to address the unique challenges faced by system delivery teams [6]. ASM entails three different categories namely; core agile development, DAD, and agility at scale [6]. These categories are presented in Figure 2.12. Core agile methods such as Scrum and XP address a portion of the SDLC and were intended for smaller co-located development teams [6]. For instance, Scrum defines a high-level life cycle and scope for development iterations [75]. XP focuses on software development activities such as continuous integration, pair programming, and refactoring [76]. However, the activities of these methods such as the Daily Scrum, requirements elicitation, and customer collaboration, etc. are a core foundation of ASM. DAD, as discussed earlier, focuses on integrating activities

of a full delivery cycle into the development of LSPs. The third category, agility at scale extends DAD by incorporating one or more scaling factors [6]. The scaling factors are discussed by Ambler (2009), as:

- Team size
- Geographical distribution
- Regulatory compliance
- Domain complexity
- Organisational distribution
- Technical complexity
- Organisational complexity
- Enterprise discipline

In order to address these scaling factors in a project, it is required to tailor the DAD activities and in some cases adopt new practices to mitigate the risk of a particular scale factor [6].

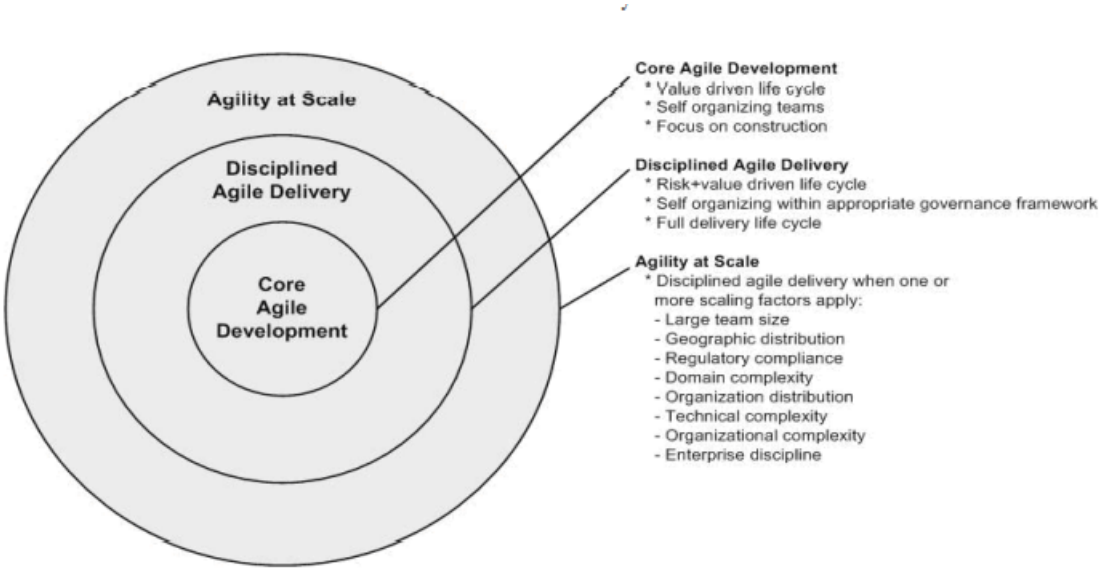


Figure 2.12: Overview of ASM. Adopted from [6].

2.4.4 Large-Scale Scrum (LeSS)

LeSS is a framework created by Craig Larman and Bas Vodde [71]. The framework handles the challenges of scaled projects using Scrum and Scrum principles [72]. LeSS operates with up to eight Scrum teams, working to achieve a common goal and collaborating with joint sprint planning, joint sprint reviews and retrospectives [72]. Similar to Scrum, each team has its own Scrum Master. However, there is a single Product Owner who oversees all Scrum teams, creates and manages the common Product Backlog [72]. The duration of the

iteration is the same for all Scrum teams. At the end of each iteration, the goal is to produce on shippable product increment [72]. LeSS does not define any new artefacts or roles in addition to those defined by Scrum [72]. A simplified version of LeSS can be found in Figure 2.13.

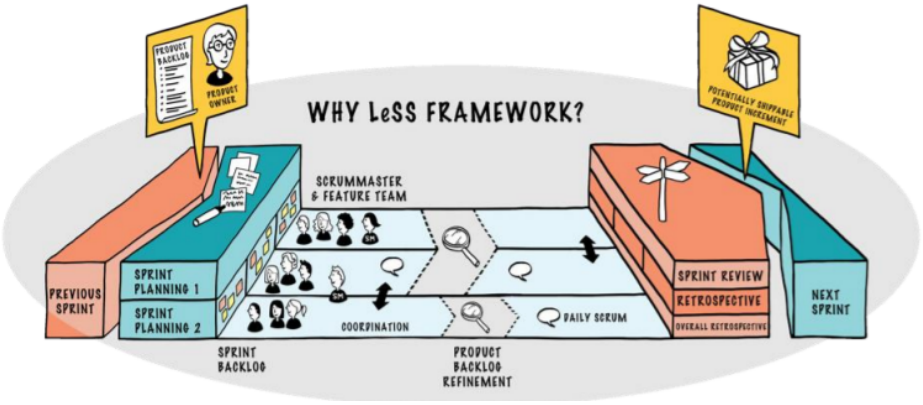


Figure 2.13: An overview of LeSS. Retrieved from [72].

2.4.5 Scrum@Scale (S@S)

In comparison with methods such as LeSS and Nexus, S@S adopts a different approach toward the scaling of agile software projects [72]. As illustrated in Figure 2.14, S@S entails two cycles and twelve key areas where specific best practices can be established in the organisation over time [72]. These areas for improvement such as backlog prioritisation, release planning, deployment, and cross-team coordination, among others, are a guide for companies to improve upon. The S@S framework does not define a specific approach to implement these key areas, as they should be tacked in a company-specific context [72].

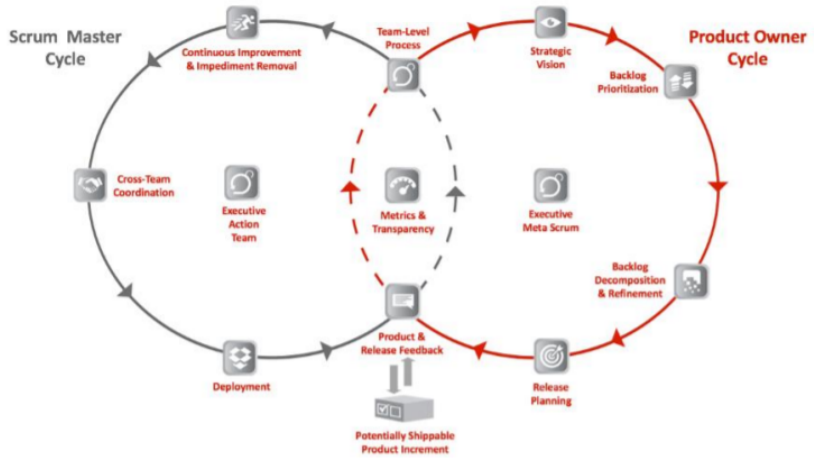


Figure 2.14: An overview of Scrum@Scale. Adopted from [72]

In comparison with Scrum, there are additional roles formed in S@S to coordinate the various teams. The Product Owners of the individual teams form a team known as MetaScrum, which is led by a Chief Product Owner (CPO) [72]. The MetaScrum team maintain a single and prioritised backlog to avoid overlap and duplication [72]. Also, a coordinated release plan and monitor metrics are decided by the MetaScrum team to ensure efficiency in the product development [72].

2.4.6 Internally Created Methods

In cases where neither agile methods nor scaled agile methods suit the needs of the organisation or specific scaling factors, internally created methods are introduced. In general, agile methods do not provide specific guidelines for large organisational practices such as the front-end and back-end process of software development [77]. The front-end process includes the approval of business cases, budgeting, and project onboarding [77]. Whereas, the back-end process involves the activities of interacting with stakeholders such as vendors and having the product ready for deployment to users [77]. An example is evident in the study of Vaidya (2014) on adapting scaling agile practices on Cambia Health Solutions [77]. Initially, they explored Scrum and Kanban on a team level. It was studied that Scrum does not give recommendations beyond the team level. However, the expectation in the Scrum method is an engaged leadership that empowers the team to be self-organising [77]. Kanban provides fewer guidelines than Scrum. In general, the two practices recommended are visualisation of the workflow and limiting the work-in-progress [77].

After a few years, Cambia Health Solutions (CHS) had more than forty agile teams, but the rest of the organisations still continued to use traditional methods [77]. Their fundamental framework to align the agile teams was SAFe [77]. However, the problem still persisted on how to address practices beyond the team level. The goal was to design a framework that adopts, combines, and modifies practices to suit their organisational context and business needs [77]. Agile practices at team levels give rise to questions concerning the organisation's design, command-and-control hierarchical management, personnel management, management practices, and the overall company culture [77]. The agile scaling frameworks they considered attempted to address some of these issues but also entailed some shortcomings [77]. SAFe is criticised as being "overtly process heavy" in the context of CHS [77, p. 10]. Having a single Product Owner who oversees all Scrum teams in LeSS results in insufficient communication with the business for the individual Scrum team as they require interaction with the business users and stakeholders on a regular basis throughout the cycle of the project [77].

CHS did not adopt a specific agile scaling framework, but rather, they embraced a combination of agile practices to suit their objectives [77]. An abstract view of the scaled agile approach taken by CHS is illustrated in Figure 2.15. Each quarterly planning defines sprints which include a 'hardening sprint' to coordinate between the component teams and account for any remaining work that could not be completed during the sprint in order to ensure a potentially shippable code as part of the entire solution [77]. The quarterly planning event is similar to the Release Planning meeting of SAFe. The difference is that not

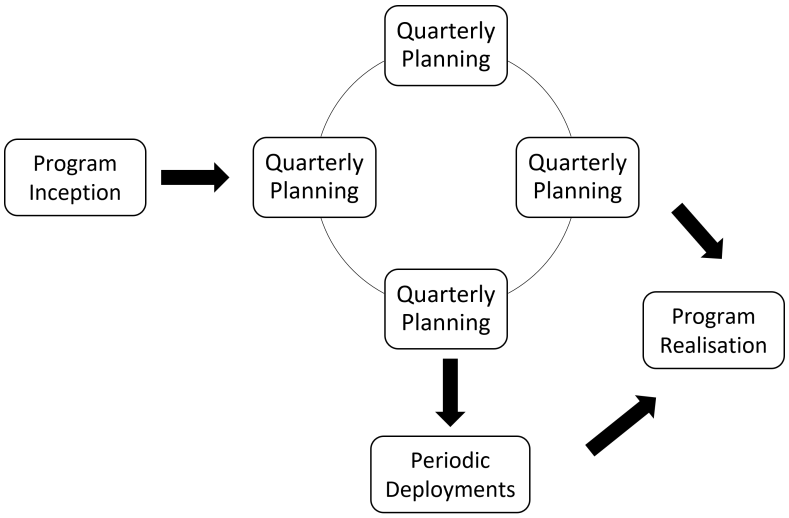


Figure 2.15: An overview of the scaled agile approach devised by CHS. Adapted from [77].

all teams attend and generally, two or three representatives (usually with seniority) from each team attend this meeting. Instead of planning a Potentially Shippable Increment as in SAFe, they define the quarterly team backlog with deliverables related to the project, non-backlog items such as enhancements, product support and other maintenance tasks [77]. Scrum-of-Scrum meetings as recommended by SAFe and LeSS are used here to coordinate and synchronise the work of the various teams [77]. Finally, Communities of Practices (CoP) is a crucial activity in the agile scaling approach of CHS. CoP refers to "a group of people who share a craft or profession" [77, p. 14]. Under the CoP practice, a Center of Excellence is formed. This is a community whereby knowledge, expertise, best practices and support are shared within the company for a specific focus area [77].

Other literature sources refer to internally created methods as hybrid methods. This approach "represent a solution that, regardless of company type and industry sector enables companies to benefit from both worlds by providing clients and managers a safe environment and developers with the demanded flexibility" [78, p. 26]. Hybrid methods combine activities from pure traditional plan-driven methods with activities from agile methods [25]. Examples of some practices that distinguish traditional methods from agile methods are as follows:

- **Planning defines the project [79].** Upfront planning in agile methods includes much fewer details compared to that required by traditional methods. The plan in traditional methods generally entails a detailed description of the tasks, resources, and time which infers cost and time estimates [79].
- **Documentation is enforced throughout the SDLC [79].** Documentation within traditional methods throughout the SDLC is a means to analyse and address potential risks in the project [79]. It entails the problem and a proposed solution, allowing stakeholders outside the team to sign off on them [79].
- **Requirements engineering without development [79].** Traditional meth-

ods often impose planning requirements which entail business analysts working outside development to engage with the business and create documents that describe the business problem [79].

Water-Scrum-Fall is an example of a hybrid method that distinguishes three steps namely: water, scrum, and fall. The 'water' step defines the requirements upfront before starting with the development of the project [79]. In some companies, this step is used to derive requirements that form the basis of a contract between the business and IT, which defines the project scope, timeline and budget [79]. Afterwards, Scrum practices such as building a cross-functional team, testing within the sprint, collaboration with business stakeholders and managing change are used in the development phase of the software project [79]. Lastly, the 'fall' aspect "means establishing gates to limit software release frequency" [79, p. 12]. Even though agile methods enforce frequent releases of the software in iterations, most organisations lack the adequate architecture to support the deployment of frequent releases [79]. Therefore, in the 'fall' phase, agile teams are required to improve the situation to the best of their ability by realising the following practices [79].

In Water-Scrum-Fall, operations and development activities should be in close collaboration [79]. Agile methods encourage the building of cross-functional and self-organising teams [5]. By building a cross-functional team with development and operations expertise, this ('fall') bottleneck can be removed [79]. Subsequently, the agile teams should include release activities such as production testing, data migration, security, and performance testing in the sprint [79]. This ensures that the team can deploy their frequent releases and make use of rapid feedback [79].

2.4.7 Comparison of scaled agile methods

In this subsection, we compare the aforementioned scaled agile methods. SAFe is distinguished by having four levels: the team, program, portfolio, and large solution level [72]. At the team level, practices from agile methods such as Scrum, XP, and Kanban are incorporated [74]. DAD also uses activities from agile methods in small iterations and integrates them into a full delivery life cycle [6]. In contrast to SAFe which focuses on delivering a portfolio of epics [74], DAD focuses on delivering a solution and not only a software, which may entail the software, hardware, and documentation [6]. ASM builds another layer upon DAD, here scaling factors such as team size, geographic distribution, compliance, etc. are used to tailor the activities of the full-delivery cycle defined by DAD [6]. Similar to the discussed methods, LeSS uses Scrum at the team level, but only Scrum [72] and no other practices from other agile methods. The teams are aligned by a single Product Owner who oversees all the Scrum teams [72]. Unlike SAFe, where new roles are explicitly mentioned, LeSS does not introduce any new roles [72]. S@S introduces a new team known as MetaScrum which consists of the Product Owners of the individual teams [72]. Unlike the other scaled agile methods, S@S does not define a specific approach, but rather, a set of best practices in the form of two cycles are presented [72]. The perspective S@S take on implementing the best practices is that they should be tackled in a company-specific context [72].

Furthermore, we compare these methods based on the aspects of principles and practices. These aspects for comparison are in line with two of the categories for comparison of scaled agile methods in a systematic literature study by Edison et al. (2021). In the study, the authors compared SAFe, LeSS, S@S, DAD, and the Spotify model from 191 primary studies across 134 organisations [28]. Before discussing the comparison, we first explain the aspects for comparison. A principal is defined as a proposition that serves as a foundation for a system [28]. It defines the grounds for making decisions throughout the processes of a method [28]. Moreover, practices are defined as customary ways of doing something which is recognised by a community as the right way to perform an activity [80]. In Tables 2.3 and 2.4, we present a summary of the comparison of the principles of SAFe, DAD, ASM, LeSS, and S@S respectively. More details on the various principles and practices of the methods can be found in [28].

Table 2.3: A summary of the comparison of scaled agile methods on principles

Method	**Principles**
SAFe	"(i) Take an economic view, (ii) Apply systems thinking, (iii) Assume variability; preserve options, (iv) Build incremental with fast, integrated learning cycles, (v) Base milestone on objective evaluation of working systems, (vi) Visualise and limit WIP, reduce batch sizes, and manage queue length, (vii) Apply cadence; synchronise with cross-domain planning, (viii) Unlock the intrinsic motivation of knowledge workers, (ix) Decentralise decision-making, (x) Organise around value" [28, p.32].
DAD	"Delight Customers, Be Awesome, Pragmatism, Context Counts, Choice is Good, Optimise Flow, Enterprise Awareness" [28, p. 32]
ASM	The principles of ASM are not explicitly stated in the literature. Rather it focuses on scaling agile methods to suit the scaling factors of the organisation and to include the management a full delivery life cycle [6].
LeSS	"(i) Large-Scale Scrum is Scrum, (ii) More with LeSS, (iii) Lean Thinking, (iv) Systems Thinking, (v) Empirical Process Control, (vi) Transparency, (vii) Continuous Improvement Towards Perfection, (viii) Customer Centric, (ix) Whole Product Focus, (x) Queueing Theory" [28, p. 32]
S@S	"Openness, Courage, Focus, Respect, and Commitment" [28, p. 32]

Table 2.4: A summary of the comparison of scaled agile methods on practices

Method	**Practices**
SAFe	"Build solutions components with high functioning Agile Release Trains, Build and integrate with a solution train, Capture and refine systems specification in solution intent, Apply multiply planning horizons, Architect for scale, modularity, reusability, and serviceability, Manage the supply chain with 'systems of systems' thinking, Apply 'continuous integration', Continually address compliance concerns" [28, p. 32].
DAD	"Scrum-based life-cycle, Lean-based life-cycle, The Continuous Delivery:Agile Lifecycle, The Continuous Delivery:Lean Lifecycle, The Exploratory (Lean Startup) Lifecycle, The Program Lifecycle for a Team of Teams" [28, p. 32]
ASM	The practices entail three levels. Core agile development: ensuring a value-driven life cycle, self-organising teams, and a focus on construction [6]. Disciplined Agile Delivery: ensuring risk and value-driven life cycle as well as a full delivery life cycle and self-organising teams with an appropriate governance framework [6]. Agility at scale: Ensuring DAD when one or more scaling factors apply to the project [6].

Table 2.4: (continued from previous page)

Method	Practices
LeSS	Sprint planning, Sprint review, Retrospective, Overall retrospective, Daily Scrum, Coordination and Integration, Communicate in code, Component communities and mentors, Scrum of Scrums, Multi-team meetings, Requirement Areas, Area Product Backlog, Area Product Owner Technical Excellence (Specification by Example Continuous Delivery, Continuous Integration, Test Automation, Acceptance Testing, Architecture and Design, Clean Code, Unit Testing, Thinking about Testing, Test-Driven Development) [28]
S@S	"the Scrum Master Cycle (Continuous Improvement and Impediment Removal, CrossTeam Coordination, and Deployment), the Product Owner Cycle (Strategic Vision, Backlog Prioritisation, Backlog Decomposition and Refinement, and Release Planning), Scaled Daily Scrum" [28, p. 32]

2.5 Software Documentation Practices

Software documentation is one of the recommended and oldest practices in software engineering, yet it is often lacking [14]. In this section, we first introduce software documentation in general. Then we discuss some of the current challenges associated with documentation. Subsequently, we discuss the documentation artefacts, documentation practices, and the tools used for documentation in ASD.

A software document is an artefact aimed at human readers which serves the purpose of communicating information on the software system [14], [81]. A document can either be presented in the format of text or in combination with visual models or code comments [82]. Documentation is an important tool for communication [81]. It serves as a medium to record and communicate decisions on the software system [83]. Another definition of software documentation is presented by Aghajani et al. (2019) as a medium that "provides developers and users with a description of what a software system does, how it operates, and how it should be used" [20, p.1199]. According to Barker (2003), software documentation is a formal writing as a hard or soft copy that supports the efficient and effective use of software in its intended environment [84].

In the context of this research, a software document refers to a written document that captures information about the software system. Whereas, the definition of documentation includes other media such as the agile artefacts discussed in Section 2.5.3. Documentation includes several software documents, which accompany the development process. An example of the use of documentation is in describing requirements, design and marketing demands, end-user manuals, and technical documentation [83]. Similar to the use of the term 'software documentation' by Ding et al. (2014), we use it to represent the document artefacts as well as the documentation activity [85].

Documentation is useful throughout the various phases of the SDLC [85]. The Software Requirements Specification and Software Architecture Document are examples of formal documentation produced in the requirements elicitation and design phases respectively [85]. In practice various forms of documentation are used in the creation of this formal documentation, most of this documentation is stored in files such as Microsoft Word documents, emails, text messages, blogs, and wikis [86].

2.5.1 Benefits of Documentation

Software documentation can be classified as process and product documentation [87]. And also, as technical and non-technical documentation [18]. We first explain these perspectives and then, we discuss the benefits of documentation.

A perspective on the classification of documentation produced in large software projects is presented by Sommerville (2005). The author distinguished process and product documentation. Process documentation 'record the process of development and maintenance' [87, p.3]. Examples of such documentation are Project Plans, Reports, and Organisational or International standards [87]. These kinds of documentation aid the management activities of the projects [87]. In turn, product documentation 'describes the product that is being developed' [87, p.3]. Product documentation distinguishes two categories which are system documentation and user documentation. System documentation 'describes the product from the point of view of the engineers developing and maintaining the system' [87, p.3]. This is similar to the technical documentation discussed in the work of Garousi et al. (2015). Whereas, user documentation 'provides a product description that is oriented towards system users' [87, p.3].

Technical documentation [18] is also referred to as internal documentation in the literature [1]. Internal documentation is used by software practitioners to help understand the system [18]. The use of documentation was briefly highlighted earlier in this chapter. In this sub-section, we dive deeper into the various merits of documentation. In a previous study, emphasis is made on the relevance of documentation for knowledge sharing in distributed and global software development projects due to the absence of face-to-face communication [1]. Survey analysis was conducted with responses from 79 agile practitioners in 8 teams and 13 countries to investigate the perceptions of the relevance of documentation. It turned out that more than half of the respondents found internal documentation (very) important but not present enough in their projects [1]. Throughout the SDLC, documentation plays a crucial role in the various phases [18]. In the design phase, design documents are prepared to visualise the software goals and architecture [18]. Subsequently, in the implementation phase, documentation aids development and testing activities [88]. Examples are the use of UML class and state diagrams to assist in unit testing and interaction diagrams being used in functional and regression testing [18]. In the maintenance phase, documentation serves as a guide to understanding the existing system [18].

Generally, documenting the decisions made during all activities of the SDLC aids the onboarding of and transfer of knowledge to new team members on the project [18]. In the absence of adequate documentation, the only reliable source of information on the system is the source code [18]. Research shows that it tends to be very time-consuming to understand the functionalities of the system by exploring the source code [89]. For large software teams, even though agile principles stress verbal communication, it is almost impossible to keep all team members up to date. Hence, the necessity of documentation to facilitate knowledge sharing in the case of large teams [1], [18], [90].

In a study conducted by Zhi et al. (2015), a meta-model was developed to classify the various benefits of documentation from existing literature [82]. This

is shown in Figure 2.16. In the meta-model, the authors present *maintenance aid*, *development aid*, *management decision aid*, and *other aid* as the benefits of documentation to software practitioners [82]. Comprehension aid is an essential aspect of maintenance and development as the software documentation needs to be properly understood before being modified [82]. Furthermore, documentation assists managers in decision-making processes such as assigning responsibilities to developers [82]. Other refers to any other benefits of documentation that cannot be categorised as maintenance, development or management decision aids such as reusing documentation [82].

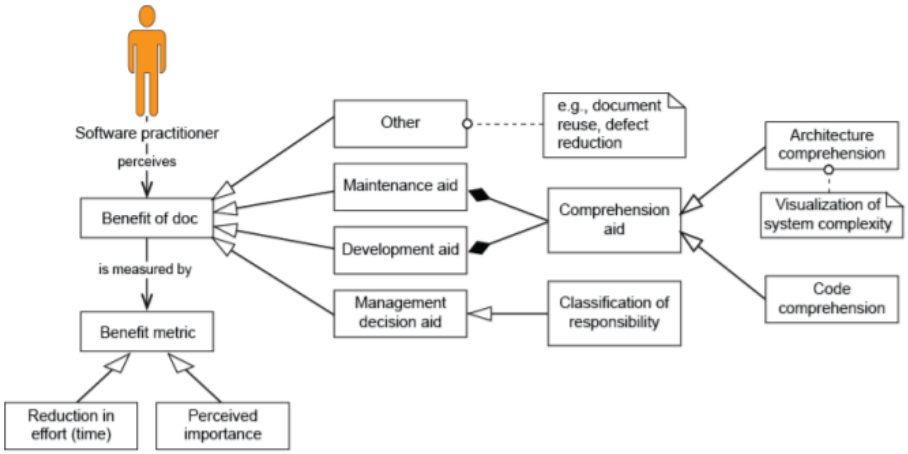


Figure 2.16: The benefits of documentation. Adopted from [82].

2.5.2 Challenges Concerning Documentation

A number of past studies have discussed the issues and challenges facing documentation. According to the observation of Aghajani et al. (2019), previous studies investigating these issues have been based on surveying and interviewing developers, which leads to a somewhat biased perspective of the challenges affecting documentation [20]. In their research, they conduct an empirical study with 878 documentation-related artefacts to investigate software documentation issues [20]. This set of 878 artefacts includes development emails, programming forum discussions, issues as well as pull requests related to software documentation [20]. In order to analyse the challenges facing documentation, it is important to understand what 'good documentation' is. In their research, Aghajani et al. (2019) identified certain criteria from their empirical data which define what good documentation should capture as well as the problems in these categories. The taxonomy of documentation issues distinguished four categories; Information Content (What), Information Content (How), Process Related, and Tool Related issues [20]. We focus on the information content and process-related issues and leave out the tool-related issues category. This is because the issues found in the documentation tools are not specifically related to the documentation, but the tool itself [20].

Information Content (What) captures the issues of what is written in the documentation [20]. Figure 2.17 illustrates a simplified version of this category of

issues. The criterion defined by Aghajani et al. (2019) with regards to the information content of good documentation is **correctness**, **completeness**, and **up-to-dateness**.

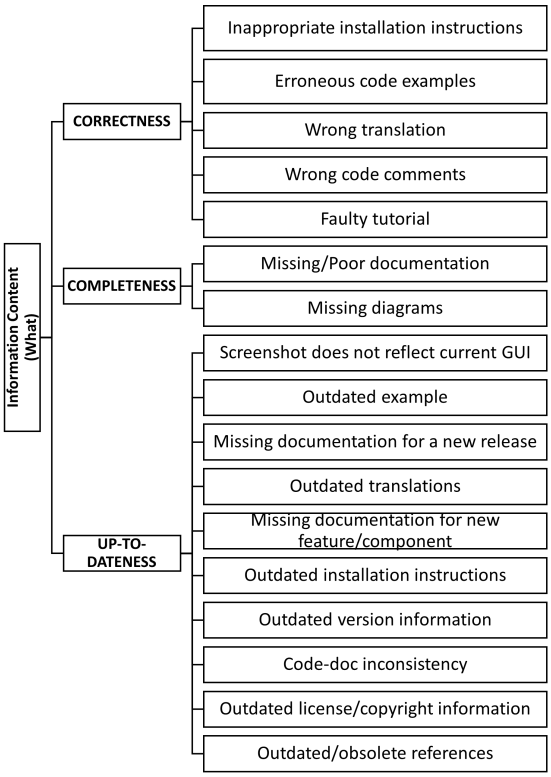


Figure 2.17: Information content (what) related issues. Adapted from [20].

Correctness ensures that the information provided in the documentation is precise and in accordance with the facts [82]. Incorrect documentation might lead to issues such as time loss, following the wrong steps in a tutorial (in the case of an end-user artefact), serious programming errors, and misunderstanding due to inaccuracy or issues such as wrong translation [20]. Moreover, *completeness* ensures that the documentation provides the information needed by stakeholders about the system or its modules to perform their tasks [82]. This criterion highlights two aspects, namely missing or poor documentation and missing diagrams. Evidence of missing definitions of ambiguous terms was identified, an example is the use of the term 'frequently' without an explanation of what it means in the context [20]. Another issue identified in this criterion was problems corresponding to missing descriptions of library components and API documentation [20]. Finally, *up-to-dateness* ensures that the documentation is in sync with the other parts of the system [20]. A major difference with the other two criteria is the information can be correct and complete before the introduction of a change [20]. One of the identified issues was inconsistency as the system's behaviour did not conform to the description in the documentation [20]. This inconsistency was a change in the code that required certain parts of the doc-

umentation to be modified [20]. The inverse is also evident in their findings, whereby code does not reflect with is described in the documentation, for example, requests by users [20]. In one of their data sources, a debate was posed as to whether the code or documentation needs to be updated in that case [20]. Also, deprecated information in the documentation is another violation of up-to-dateness [20], which can result in misleading information.

The second category introduced by Aghajani et al. (2019) is *Information Content (How)*. Figure 2.18 represents a simplified version of this category of issues. It defines **maintainability**, **readability**, **usability**, and **usefulness** as subcriteria to discuss problems relating to the writing style and organisation of documentation [20].

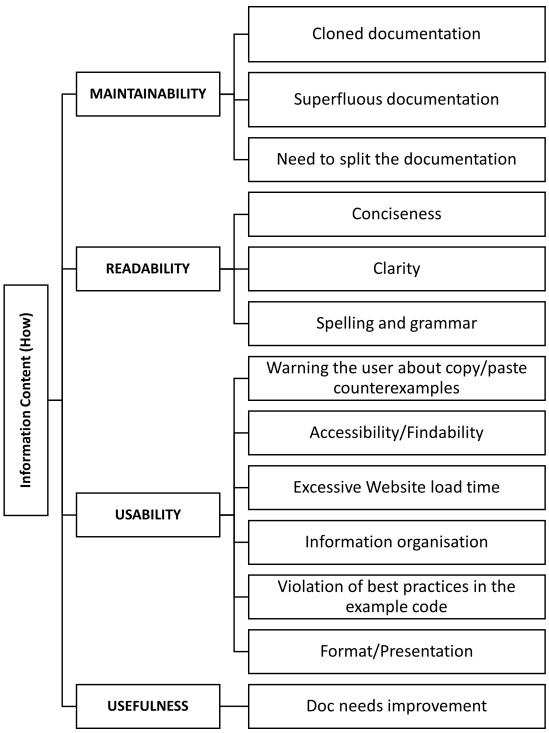


Figure 2.18: Information content (how) related issues. Adapted from [20].

"*Usability* of documentation refers to the degree to which it can be used by readers to achieve their objectives effectively" [20, p. 1205]. Half of the issues found by Aghajani et al. (2019) related to the ease of finding information. There were questions about missing parts of the documentation. Additionally, information organisation was the second most common concern in this category [20]. It refers to the degree to how intuitively and clearly information is organised [82]. Moreover, the availability of documentation was also evident as an issue in their analysed artefacts. A minor issue was poor or inconsistent formatting such as missing headings, and template structure. However, they are seen as minor issues because they are perceived as "not really a barrier for using documentation" [20, p. 1205]. *Maintainability* refers to the degree of ease to

apply changes to documentation [20]. A concern in this criterion is duplicated content, which impedes the effective maintenance of documentation. Additionally, the evidence of superfluous files was seen as a potential cause of confusion for the stakeholders [20]. Not all information is needed by each stakeholder and in turn, large documents are difficult to maintain. *Readability*, on the other hand, defines the degree of ease to read a document. Issues related to lack of clarity and typos were identified as the most common problems with regard to this category [20]. Lastly, *usefulness* defines whether the documentation "is of practical use to its readers" [20, p. 1205]. Each stakeholder has different needs and that affects the degree of usefulness of a particular artefact. Feedback from various stakeholders is therefore crucial to ensure useful documentation [20].

The last category presented by Aghajani et al. (2019) entails *Process Related* issues, as shown in Figure 2.19. This category defines **internationalisation**, **traceability**, **development issues caused by documentation**, **contributing to documentation**, and **doc-generator configuration** as sub-categories of issues related to the documentation process.

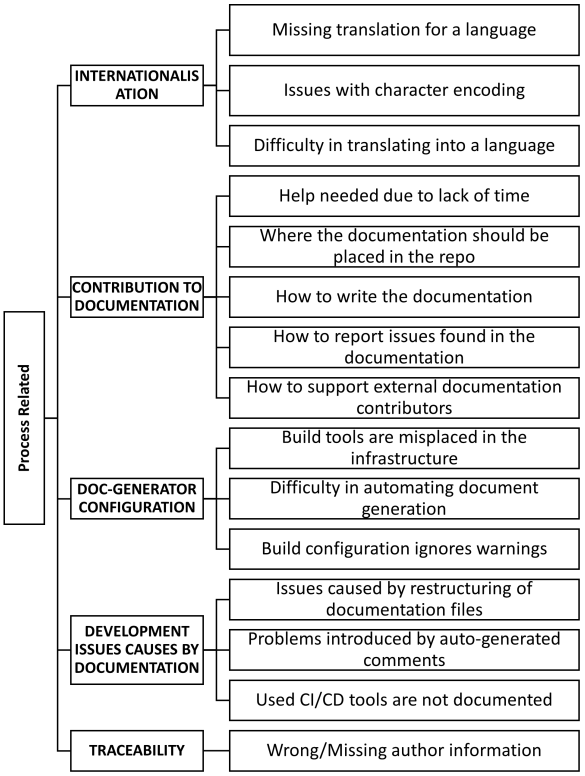


Figure 2.19: Process related issues. Adapted from [20].

Firstly, *internationalisation* discusses issues related to the translation process, the need to review translated documents as well as the problems caused by character encoding [20]. Furthermore, *traceability* describes the issues relating to the ability to track changes made to the documentation. A solution to keeping track of changes to documentation is using version control systems, however, this is

a challenge for documentation in binary format or in a database [20]. Moreover, development issues caused by documentation entail the "unwanted effects of documentation on the development process" [20, p. 1207]. An example is the effect of auto-generated documentation on the code itself [20]. Additionally, *contribution to documentation* highlights the issues that contributors with regard to writing documentation [20]. The scarcity of well-explained guidelines and sometimes documentation templates are common issues within this subcategory. The lack of knowledge about best practices to write documentation are also an evident issue. At last, *doc-generator configuration* issues related to documentation generators [20]. These are caused by infrastructure or configuration errors resulting in warnings that affect the complete generation of documentation.

To conclude, the challenges facing documentation were categorised by Aghajani et al. (2019). Two potential future research directions on documentation practices were suggested that may be relevant to this research. It was advised that future research should optimise the documentation processes and answer fundamental research questions, for example, *what makes a good contributors guide?* [20].

2.5.3 Artefacts in ASD

An artefact is defined by Wagenaar et al. (2015) as "a tangible deliverable produced during software development" [91, p. 2]. In this context, tangible was defined as "being easily seen or recognised rather than being restricted to only being touched or felt, thus including materials in both physical and electronic format" [91, p. 2]. Artefacts in ASD include architecture, requirements, and design artefacts [92]. In turn, artefacts serve as a means of communication and knowledge transfer in software development [19].

Communication is a fundamental aspect of all agile methods [91]. The Agile Manifesto emphasises that "The most efficient and effective method of conveying information to and within a development team is face-to-face conversation." [5]. Contrary to face-to-face communication is documentation by means of artefacts [91]. The practice of explicit documentation is not encouraged in ASD, which conforms to the Agile principle of having working software over extensive documentation [5]. Although ASD methods are described as lightweight, there are many artefacts scattered throughout the entire ecosystem of tools as opposed to being documented in a single self-contained document [17]. Therefore, "*'working agile'* and *'using artefacts'* are no contradictory terms" [91, p. 135]. From Figure 2.7, it can be observed that the primary artefacts of Scrum are the Product Backlog, Sprint Backlog and the Software Product Increment. In Kanban, the main artefact is the Kanban Board as discussed earlier. And XP utilises story cards, task lists, customer acceptance tests, Class-Responsibility-Collaboration cards, acceptance tests, and visible wall graphs as artefacts [76].

Wagenaar et al. (2015) performed a case study on three organisations using Scrum, identified, and compiled the artefacts used in ASD from a Scrum perspective. Prior to conducting the case study, the authors identified an artefact model by Kuhrmann et al. (2013) as the basis for expansion in their case study. This Agile Scrum artefact model (Figure 2.20) was derived from a systematic

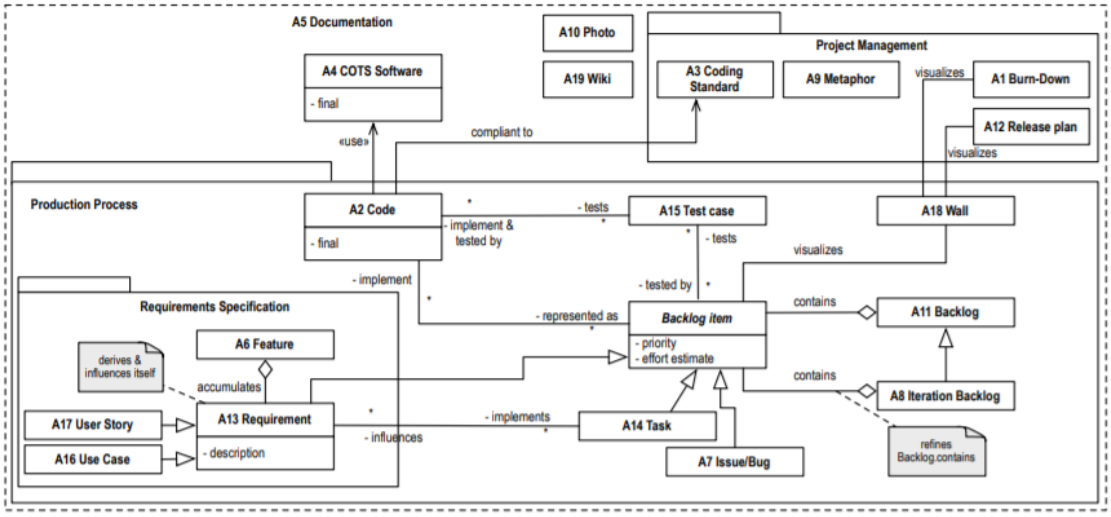


Figure 2.20: Initial Scrum artefact model for agile methods. Adopted from [93].

literature review and consists of four parts; project management, requirements specification, production process, and other documentation parts [93]. The artefacts used in Scrum development are more than the core Scrum artefacts themselves. The other artefacts can be categorised into non-scrum, product and process artefacts [91]. These categories were also applied to the results of their case study and the artefact model was extended with design, test, and release-related material. Figure 2.21 presents the extended model of artefacts in Scrum.

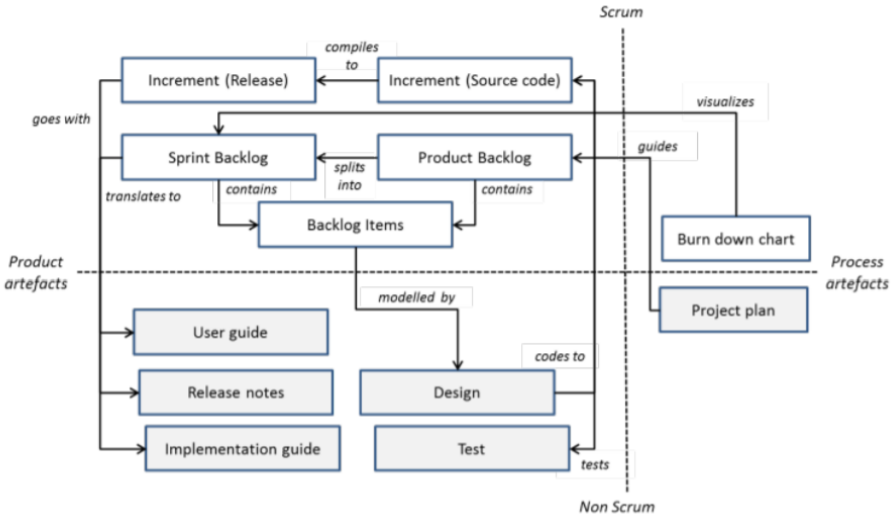


Figure 2.21: Extended Scrum artefact model for agile methods. Adopted from [91].

A subsequent study on artefacts in ASD built upon the findings of Wagenaar et al. (2015). In the study, the rationales behind the use of agile artefacts were investigated in 19 agile teams. These organisations were software-producing organisations and were involved in software product management. The findings revealed 55 artefacts with confirmed the previous results on artefacts in ASD [25]. Five rationales for the use of artefacts in ASD (Figure 2.22) were identified as agility, governance, internal communication, quality assurance, and external follow-up rationale.

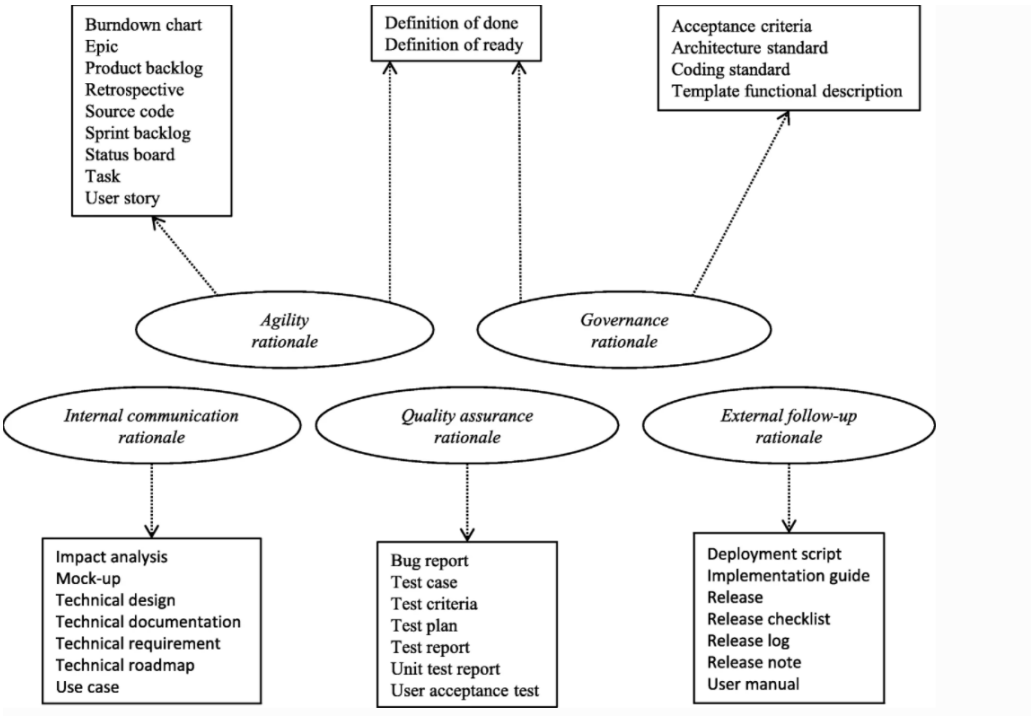


Figure 2.22: Categories of rationales for the usage of agile artefacts. Adopted from [25]

Agility represents the rationale behind the usage of artefacts due to the adoption of an ASD method [25]. For instance, the use of a Product and Sprint Backlog in Scrum. *Governance* defines the rationale for which artefacts the agile team decides to use. This rationale may be influenced by the organisation or imposed externally [25]. Examples are the acceptance criteria, the definition of done (DoD), and the definition of ready (DoR). Agile teams value *internal communication*, hence artefacts such as a design flow, functional design, and technical documentation aid communication and transfer of knowledge among team members. Moreover, *quality assurance* refers to the rationale of using artefacts to ensure quality in the project, such as a test plan, test report, and user acceptance tests [25]. Lastly, the *external follow-up* rationale relates to the usage of post-development artefacts which are not of so much use to the agile team itself but rather for external parties outside the agile team such as the customer, or another part of the organisation [25].

2.5.4 Documentation Practices in ASD

A practice is defined as an activity that is regularly conducted [17]. Previous literature has confirmed the absence of practices or tooling that can be used for documentation purposes and rather, documentation information is distributed across various tools in the eco-system of the software development team [17]. A review of tools within this domain can be found in Section 2.5.6. In this sub-section, we discuss the known documentation practices in literature as well as the perceptions of documentation in agile teams.

The most evident knowledge-sharing practice in ASD is verbal communication [17]. As mentioned earlier, this is one of the principles of agile methods stated in the Agile Manifesto [5]. It is more effective in aiding mutual understanding compared to written documentation [17]. Often verbal communication in agile teams is supported by sketches, informal drawings [17], and flow diagrams. However, ASD also entails formal communication which blends both traditional and agile artefacts [94]. User stories are widely used as an artefact to communicate requirements [17]. In addition to artefacts created for documentation purposes such as knowledge transfer, artefacts are also created as part of the development process that can serve as a type of documentation artefact [17]. An example is executable specifications of the to-be system, where tests or behavioural flow are defined in an executable format in the case of TDD and Behavioural Driven Development (BDD), respectively [17].

Although the emphasis is made on verbal communication in agile methods, there are various artefacts used in ASD with different rationales as illustrated in Figure 2.22. Knowledge sharing without documentation artefacts and solely based on face-to-face communication is likely to result in issues such as undocumented knowledge gaps, and inadequate architecture [16]. Artefacts in ASD potentially lose their effectiveness if the reader lacks a shared and overall understanding of the project [16]. Also, valuable knowledge about the software is likely to evaporate with time due to employee turnover [1]. In a research conducted by Sharp et al. (2009), empirical data was used from six teams who are mature in using XP on the topic of investigating the role of two artefacts (the story card and the Wall) in ASD [95]. With the term 'mature', they refer to a team that has used XP for more than a year and have successfully produced software [95]. They found that the artefacts in agile methods lack detailed information about the system being developed [95]. By a means of questionnaires, Stettina and Heijstek (2011) captured the perceptions of seventy-nine ASD practitioners with regard to the documentation in their projects. They found that agile practitioners admit that too little documentation is available in their projects and they spend too little time on writing documentation [1].

The shift from traditional software engineering methods to agile methods has caused companies and clients to still use software documents [83]. This as a result imposes the external follow-up rationale for the agile team. If the documentation is of no use to the team, then the motivation is rather external. One of the main reasons for the lack of enough documentation can be attributed to the lack of developers' motivation [83]. The process of writing documentation is perceived as a burdensome intrusive side-task and it is usually submitted at the end [15]. Developers tend to focus on the deadline at hand to deliver

working software and there is rarely enough time to document the decisions and knowledge after the project has been delivered [16]. In an exploratory study conducted by Shmerlin et al. (2015) on the motivation of developers toward writing code, the following findings were concluded. Documentation is perceived as a tedious task, for instance explaining how a problem was solved. Also, it was seen as a difficult task to be formal and write down the necessary decisions. Moreover, it was perceived as an intrusive task interrupting the flow of coding. Lastly, due to time constraints documentation tends to have the least priority and it tends to be neglected due to time constraints [83].

2.5.5 Guidelines on Agile Documentation

In Section 2.4.2, we introduced Disciplined Agile Delivery (DAD) as one of the approaches to scaled agile development, as a method that focuses on the full delivery life cycle of a software project. The modelling and documentation practices in DAD are based on the principles of Agile Modelling (AM) [96]. AM "is a collection of practices, guided by principles and values, for software professionals to apply on a day-to-day basis. [..] it does not define detailed procedures for how to create a given type of model, instead, it provides advice for how to be effective as a modeler" [97, p. 8]. It is usually combined with other methods such as Scrum, XP, and DSDM, to enhance the processes with a focus on modelling and documentation [97]. The term 'Agile Documentation' is introduced by Ambler (2002), as a lean and minimal documentation artefact which is "good enough in the eyes of the beholder" [97, p. 155]. With this concept, we discuss the strategies for ensuring agile documentation within AM by Ambler (2002).

- *Focus on the customer(s)* [97, p. 163]. Acquiring the stakeholders' requirements for documentation and negotiating with them for a minimal subset of documentation to ensure lean documentation [97].
- *Keep it just simple enough, but not too simple* [97, p. 163]. Start simple and build upon the documentation if necessary. To investigate the extent to which documentation should be enough, it is crucial to identify how the target stakeholders intend to use the documentation as well as the rationale of use [97].
- *The customer determines sufficiency* [97, p. 163]. Sufficiency is achieved if the documentation satisfies the intended rationales of use. To ensure that, the writer should validate whether the documentation provides value to the stakeholders [97].
- *Document with a purpose* [97, p. 164]. If a documentation artefact contributes significantly to the overall goal of the project, then it is worth creating it [97].
- *Prefer other forms of communication [over] documentation* [97, p. 164]. Even though documentation facilitates knowledge transfer, there are other alternative media of communication to consider [97].
- *Put the documentation in the most appropriate place* [97, p. 164]. Where documentation should be kept and how it should be organised depends on the needs of the stakeholders concerning that information [97].
- *Wait for what you are documenting to stabilize* [97, p. 164]. "Delay the creation of all documents as late as possible, creating them just before you need them" [97, p. 164].

- *Display models publicly* [97, p. 164]. When information about the project is displayed publicly, the transfer of information is promoted. Thereby, reducing the need for detailed documentation as the project stakeholders gain more insights into the project via the shared information [97].
- *Start with models you actually keep current* [97, p. 165]. Focus on creating models that are less likely to get outdated. Models that keep getting outdated are of no value to document [97].
- *Require people to justify documentation requests* [97, p. 165]. Inform stakeholders about what is involved in making and updating the document as well as asking the need and the justification for the need of a particular documentation artefact [97].
- *Write the fewest documents with least overlap* [97, p. 165]. A recommendation is to split large documentation artefacts into smaller parts with the least overlap, this improves the ease of maintainability [97].
- *Get someone with writing experience* [97, p. 165]. "Technical writers bring a lot to the table when it comes time to writing documentation because they know how to organize and present information effectively" [97, p. 165]. Or an alternative is to provide training to the team members on how to approach agile documentation [97].
- *Make it easy to remember the fundamentals* [97, p. 166]. Amber (2002) recommends that these guidelines should be made publicly visible for the team members to serve as a reminder of how to "take an agile approach to documentation" [97, p. 166].

However, these guidelines are rather abstract for application in practice and they are mainly channelled towards documentation created for external purposes. Also, in Section 2.5.3, we discussed the rationales for use of documentation. A factor driving the use of documentation is not only external follow-up, but also, agility, governance, internal communication, and quality assurance [25]. Therefore these guidelines may not be suitable to apply to ASD projects to address the challenges found in documentation, but are useful to provide insights into how the literature perceives agile documentation.

2.5.6 Tools for Documentation in ASD

Documentation artefacts are scattered across the entire eco-systems of tools within an ASD project and there is hardly a single source of truth [17]. As discussed in Section 2.4, scaling factors such as organisational size and governance drive the adoption and adaption of agile methods in large software projects. In a systematic mapping literature study by Theunissen et al. (2022), the developments in documentation and tools in Continuous Software Development (CSD) were discussed. CSD is an overarching term for the combination of Lean, ASD, and DevOps to cover the activities for the entire SDLC of a product [17]. Since it is a relatively new paper to the date of writing, we are convinced that it covers an updated evidence of the use of tools in agile methods. The authors found that the tools used in CSD entail information on the source code, tests, deployment, and the quality of the software [17]. The categories of tools discussed by Theunissen et al. (2022) are summarised in Table 2.5.

Table 2.5: Categories of tools in CSD. Summarised from [17].

Category	**Sub-category**	**Example(s)**
Development	Requirements management tools	Blueprint, RequirementONE
	Integrated Development Environments (IDEs)	Eclipse, IntelliJ
	Agile Management tools	JIRA, Agile Bench
	Development Analytics tools	SonarQube, Metrixware
	Repositories	GitLab
	Development community	StackShare, StackOverflow
Test	Quality attributes requirements test tools (performance, reliability, and security)	JMeter, SmartStorm
	Functional Automation tools	Cucumber, Appium
	Continuous testing tools	test.ai, Buildbot
	Service Virtualisation testing tools	Smartbear, Parasoft
Deployment	App Automation tools	Ansible, Puppet, Chef
	Continuous Integration/Continuous Delivery (CI/CD)	CircleCI, Jenkins
Service execution	Cloud/Container Orchestration/-Management	Docker, Mesos
Monitoring	Monitoring and Management tools	DataDog, RunDeck
Security	Container Security	AppArmor, Cloud Insights
	Application Security	Threat Stack, HyTrust
	DevSecOps	Cigital, CheckMarx
API management	API management tools	RapidAPI, OpenAPI
	API directories	ProgrammableWeb

The type of tools used determines the documentation information and artefacts produced and used in the agile team. "The amount of structure of the information is strongly related to the tool" [17, p. 13]. For requirements management tools, information such as stakeholders' concerns, risks, and constraints are presented as specifications which can be informal such as high-level user stories or formal and concrete such as detailed use cases with pre-and post-conditions [17]. Moreover, agile management tools such as JIRA and Agile Bench provide support for developers to find information about requirements, tasks, progress, planned and achieved goals for the iterations, as well as the traceability between requirements, tasks and code [17].

2.6 Conclusion

Throughout this section, we explored the background knowledge essential for this research. We discussed the transition of software development methods from plan-driven approaches to adaptive and iterative-driven approaches. ASD approaches have gained popularity over the past two decades. Requirements engineering activities, such as elicitation techniques, reside in ASD methods. Although agile RE practices address several challenges such as communication gaps and unreliable requirements specifications in traditional RE, they also pose new challenges such as minimal documentation. ASD methods were originally intended for use by small and co-located teams. However, due to its benefits such as adaptability and close-customer collaboration, the approach became increasingly embraced in LSOs. To address the issues of scaling agile methods, methods such as SAFe, DAD, etc. have been proposed. We see the different

adoption of scaled agile methods in the literature with various processes, roles, and use of tools. This serves as a basis for the design of our case study.

The risk of insufficient documentation in scaled-agile projects is higher than in smaller projects, as it may significantly impact the cost and timeline of the project. Ensuring the quality of documentation has been a renowned problem in Software Engineering even before the adoption of ASD. In the literature, several attempts have been made to define what makes a good documentation artefact. We adopt the taxonomy defined by Aghajani et al. (2019), as it contained most of the earlier defined aspects of good documentation. Out of the four defined categories, we focus on the Information Content (What), Information Content (How), and Process Related quality aspects of documentation. We apply these categories to identify the strengths and issues of documentation in case studies. Lastly, the feedback cycles identified in Section 2.2.4 is a potential aspect of the treatment design.

Chapter 3

Research Methods

We discuss the objective of this research using the goal-oriented approach of the Goal-Question-Metric (GQM) model in Section 3.1. Then, we introduce the design science approach which we adopted for this research, in combination with a multiple-case study. A discussion of this approach is found in Section 3.2. Finally, we present the data collection techniques and data analysis approaches in Sections 3.5 and 3.6, respectively.

3.1 Research objective

The goal of this research is presented using the Goal-Question-Metric (GQM) template by Basili et al. (1994). The GQM approach is based on declaring the objectives of the research, linking those objectives to the research to operationalise the goals, and providing a framework for understanding the data in relation to the stated objectives [98]. The GQM perspective is based on the theory that all measurements should be goal-oriented [98]. Hence, the goal of this research is to:
Analyse *the requirements engineering practices in scaled-ASD,*
for the purpose *of evaluation*
with respect *to their impact on documentation practices and artefacts*
from the point of view *of the agile project teams*
in the context *of multi-team projects.*

The GQM model distinguishes three levels namely: *conceptual*, *operational* and *quantitative* level. We explain these levels briefly.

- **Conceptual level**. Goals are defined for an object for a variety of reasons and from a specific viewpoint [98]. Examples of objects used for measurement in software engineering research are products such as artefacts and deliverables, processes such as designing and testing, and resources such as personnel, hardware and software [98].
- **Operational level**. A set of questions are identified to define how a specific goal will be performed on the object of measurement [98].

- **Quantitative level**. This level identifies the metrics to answer every question quantitatively [98].

Even though the metric is intended for quantitative measurements, the goal-oriented concept of the GQM is applied in qualitative studies such as [99], [100]. For example, Fuggetta et al. (1998), studied the application of GQM in an industrial software setting [99]. Their questions included ones that are answered by qualitative metrics such as the knowledge of the testing team on the application domain, the understandability of requirements, the origin of a software failure, and differences across certain platforms, among others [99]. Also, in a qualitative study by Machado et al. (2021), the perceptions of practitioners during a transition from a traditional to an agile process model were studied using semi-structured interviews [100]. They used the GQM to formulate the interview questions and identified the perceived benefits of the transition to ASD, the challenges faced as well as solutions to those challenges [100].

Our main research question is divided into seven sub-questions as introduced in Section 1.2. For each sub-question, we operationalise the goal using a GQM model. Figure 3.1 illustrates the goal of investigating how multi-team software projects in Large Software Organisations adopt agile methods. Here, we are interested in the type of *roles*, *processes*, and *tools* involved in the scaled agile method of the project in question.

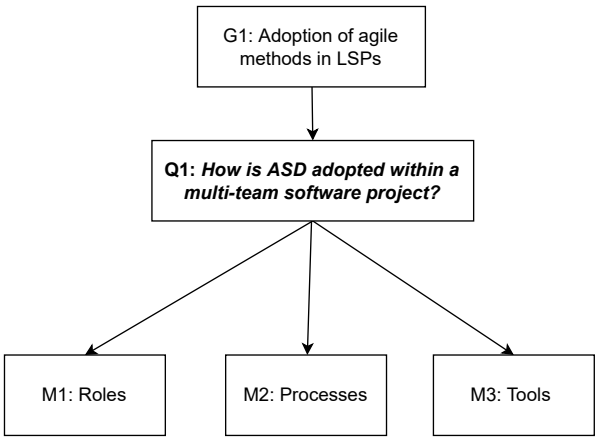


Figure 3.1: A GQM model of SRQ1

Subsequently, with the insights from the adoption of agile methods in the multi-team software project, the evidence of requirements engineering practices are studied and categorised into the phases of RE by Dieste (2009). These phases are *elicitation*, *analysis*, *specification*, *validation*, and *requirements management* [50]. These requirements engineering phases are discussed in Section 2.3. Figure 3.2 presents an overview of this categorisation as metrics to answer the second sub-research question.

The scope of software documentation is broad. Therefore, we focus on the documentation artefacts which are used in the RE practices identified from the previous research question. Hence, the goal of the third sub-research question is

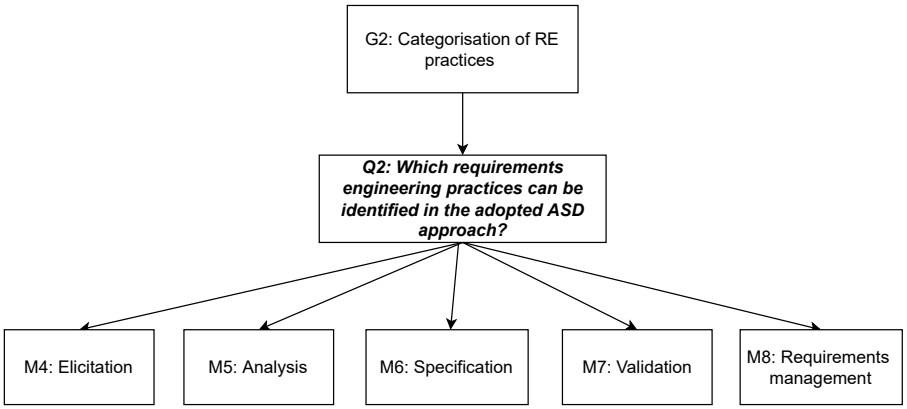


Figure 3.2: A GQM model of SRQ2

to identify and categorise the documentation artefacts into the categories introduced by Wagenaar et al. (2015) in Section 2.5.3, as *product artefacts*, *process artefacts*, *core agile method artefacts (scrum artefacts)*, and *non-agile method artefacts (non-scrum artefacts)* [91]. The GQM model of this sub-question is illustrated in Figure 3.3.

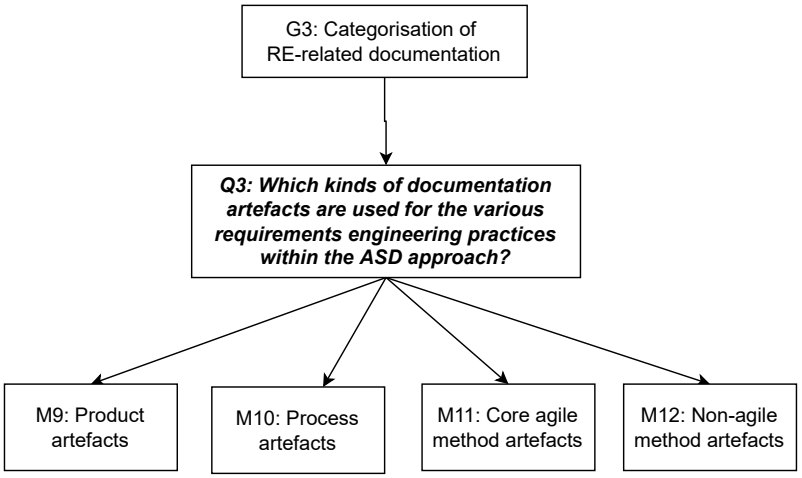


Figure 3.3: A GQM model of SRQ3

After identifying and categorising documentation related to the RE practices, the fourth sub-question analyses the strengths and weaknesses of the documentation found. Figure 3.4 presents an overview of the GQM model of this sub-question. The findings will be measured according to the categorisation of documentation issues presented by Aghajani et al. (2019), as elaborated in Section 2.5.2. The categories of good documentation were introduced as *Information Content (What)*, *Information Content (How)*, and *Process Related* categories [20]. These categories are also used to analyse the impact of requirements change on the documentation as shown in Figure 3.5.

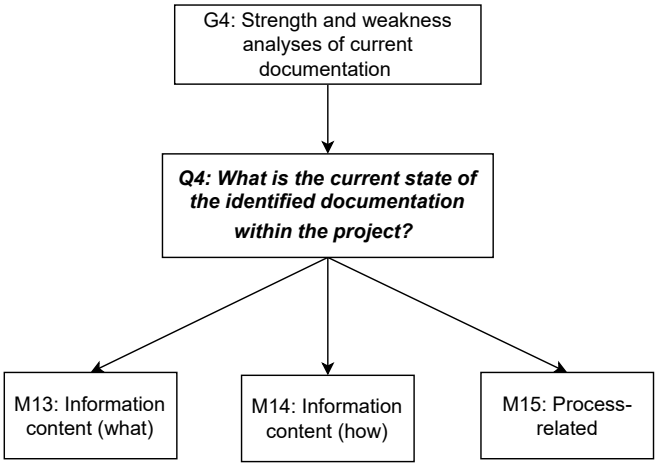


Figure 3.4: A GQM model of SRQ4

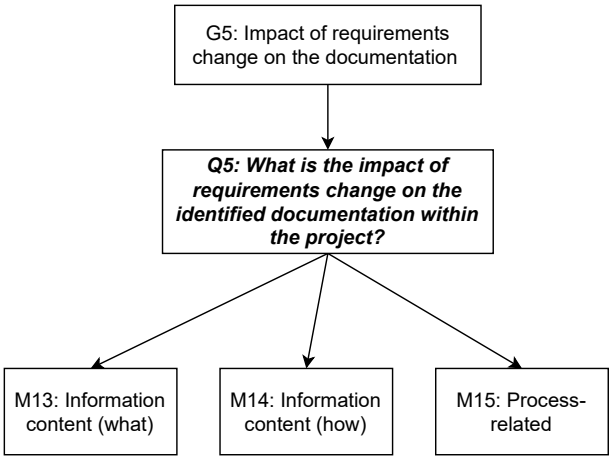


Figure 3.5: A GQM model of SRQ5

The potential aspects for improvement identified in the documentation with regard to what information is presented, how it is presented, and the process of writing documentation, are used as requirements for the framework. The goal of the framework is to address the issues found in the documentation and improve documentation practices in multi-team software projects. Hence, Figure 3.6 presents the GQM model of the sixth sub-question. The fitness of the designed framework is measured by how well it satisfies the requirements.

Finally, the designed framework is evaluated by practitioners. An experiment is conducted to study the proposed framework in two multi-team ASD projects. We conduct a literature study on what variables can be used to evaluate the framework to improve documentation practices. We adopt the research model of Green et al. (2005) to evaluate our framework, where the authors studied the impacts of quality and productivity perceptions on the use of software process improvement innovations [101]. In their study, they adopted the variables: *ease*

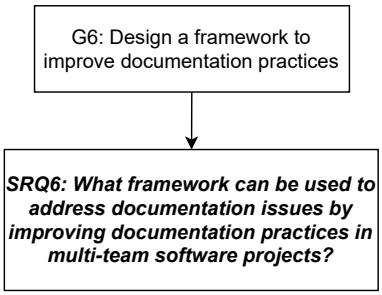


Figure 3.6: A GQM model of SRQ6

of use, *quality*, *productivity*, *usefulness*, and *use*. We adopt these variables to evaluate the effectiveness of our framework in practice. Additionally, we set up a validation survey at the end of the experiment to gather perceptions on the adoption of the framework based on the aforementioned variables. Figure 3.7 highlights the corresponding GQM model of the seventh sub-question.

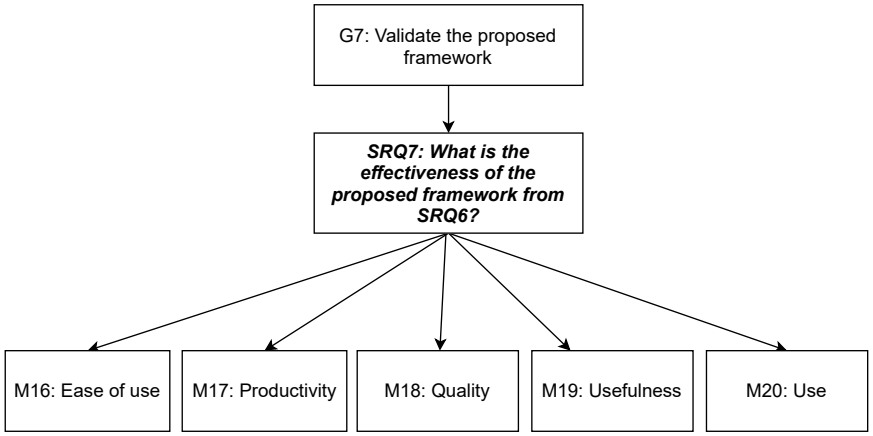


Figure 3.7: A GQM model of SRQ7

3.2 A Design Science Approach

For this research, design science is adopted and incorporated with a multiple-case study for the Problem Investigation phase. The multiple-case study enables us to study the organisations and investigate the issues facing documentation in multi-team software projects. To define a multiple-case study, we introduce the definition of a case study. Yin (2018) defined a case study as "an empirical method that investigates a contemporary phenomenon (the "case") in depth and within its real-world context, especially when the boundaries between phenomenon and context may not be clearly evident" [22, p. 15]. A multiple-case study is a case study organised around two or more case studies [22]. The design of a multiple case study must follow an analogous logic such that each case must be carefully selected to ensure that literal replication or theoretical replication

is the aim of the individual case studies [22]. The terms literal and theoretical replication imply the prediction of similar results and the prediction of contrasting results but for expected reasons, respectively [22]. Studying more than one case allows the derivation of findings to draw stronger analytical conclusions by confirming and potentially revising the initial theory based on the findings if necessary [22]. We use this multiple-case study method to answer our first four sub-questions (see Section 1.2). With the selected cases, we study the context of each case, how agile methods are adopted, which RE practices are evident, which documentation artefacts are used within the project, and what is the quality of the documentation artefacts.

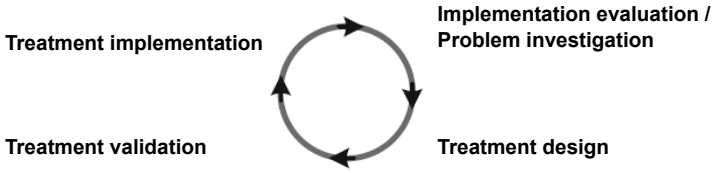


Figure 3.8: The engineering cycle. Retrieved from [21, p. 28].

The findings from the multiple-case study serve as input for the Problem Investigation phase of the design science method. Design science is a research method mainly adopted in software engineering, information systems and computer science research [21]. Design science is the design and investigation of artefacts in context [21, p. 5]. The artefacts to be studied should be designed to interact with a problem context in order to improve a problem in that context [21]. The design cycle consists of the *Problem Investigation*, *Treatment Design*, and *Treatment Validation* phases [21]. It is a subset of the engineering cycle (Figure 3.8), which consists of the Problem Investigation, Treatment Design, Treatment Validation, Treatment Implementation, and Implementation Evaluation phases [21]. The design cycle is used for this research as "design science research projects do not perform the entire engineering cycle but are restricted to the design cycle" [21, p. 30]. The Treatment Design phase is used to identify the requirements and possible treatments [21] to answer the sixth sub-research question. Moreover, the Treatment Validation phase ensures that the proposed treatment satisfies the requirements and is suitable for its context [21]. Hence, the seventh sub-question is answered using the treatment validation phase. Implementation and implementation evaluation are activities that take place outside the research, where the recommendations from the research are implemented in their original context [21].

3.3 Research Context

The setting of this research is multi-team software projects in Large Software Organisations (LSOs). The context of multi-team software projects refers to projects with **at least two teams** [102]. Each agile team should have at least five members. We choose a minimal threshold of five team members because the ideal size of teams in ASD in the literature is approximately five to nine members [103]. There is no limitation to the type of agile method being used, as in Section 2.4 it is apparent that agile methods are implemented variably

across large software organisations. Additionally, the adopted agile approach may have a different perspective on documentation. XP for instance focuses on pair-programming as a core aspect of communication [41]. Scrum, on the other hand, focuses on delivering a working iteration of the software product with close customer collaboration [42]. In scaled agile methods such as SAFe, there are additional documentation artefacts such as a shared portfolio, a local product backlog, a roadmap, and a release plan [74]. Furthermore, we aim for a variance in the type of industry or clientele range for the software projects to be able to generalise our results across large software companies.

3.4 Case Selection

The cases for this study are selected using purposive sampling, a sampling method "where members conform to certain criteria for selection" [104, p. 3511]. Therefore, the following selection criteria are defined. As discussed in detail in Section 3.5, semi-structured interviews are one of the main data collection methods in this research. The first criterion states that *the participating organisation should be willing to participate in the interviews*, allowing at most five hours for interviews per project team. Additionally, the data on documentation in ASD will not be complete if we only rely on interviews without evidence analysis. Therefore, *the second criterion ensures the willingness of the participating projects to share documentation artefacts* related to the project such as the product backlog, user stories, requirements specifications, design, and test documentation. Figure 2.22 shows an example of documentation expected to find in these project teams as well as the rationale of use. Since the seventh sub-question concerns validation of the proposed framework, *the willingness of the project team to participate in the validation activities is preferred but not mandatory.*

Using the Key Informant Technique (KIT) (Marshall, 1996), experts will be selected for the semi-structured interviews as the key informant. A key informant must possess these characteristics: knowledge, willingness, communicability, impartiality, as well as having a role with seniority in the project team [105]. KIT is a participant selection technique originally used in ethnographic research to ensure the reliability of obtaining an expert source of information [105]. The experts we aim to select from the project teams are the senior or managerial roles such as project manager, scrum master, lead developer, lead architect, lead analyst, and lead tester. As mentioned in Section 3.3, *a selected project should contain at least two agile teams with at least five members per team*, this should provide some insight into how different teams in the same project communicate aside from verbal communication. Since the context of the research is large software projects in The Netherlands, it is useful to state a selection criterion that the participant should be willing to communicate in English. Microsoft Stream will be used to automatically transcribe the recordings, as the tool is available to the researcher but the transcription feature for Dutch is not yet available in this tool.

3.4.1 Overview of selected cases

In this section, we introduce the cases to be studied in this research. We found three large organisations that were willing to participate in this research. However, only two of these companies had multi-team software projects. Therefore, the case study is based on these companies, referred to as C1 and C2 due to confidentiality. C1 is an IT consultancy firm with their main focus on digitalisation in the public sector. This company has about 4000 employees worldwide, having offices in Europe and Asia. For this research, we were assigned a multi-team project within The Netherlands referred to as P1. The project develops a web-based solution for resource planning in the educational sector.

C2, on the other hand, is a civil engineering firm in The Netherlands with about 1000 employees, that delivers designs and advice for construction projects within the country. For this research, we were assigned to the first and only multi-team software project at the time of writing. This project is referred to as P2, and it is found in the Digital Transformation department of C2. The aim of this department is to optimise the daily work of the civil engineers in order to increase efficiency within the company. Hence, the project develops a solution for performing calculations and optimisation of design activities for construction projects. The solution is used by the civil engineers within the company. In table 3.1, we provide an overview of the companies, projects, and team information.

Table 3.1: An overview of the selected cases.

Company ID	**Project ID**	**Project duration (on March 2022)**	**Team ID**	**Team size**
C1	P1	$2\frac{1}{2}$ years	P1T1	8
			P1T2	6
C2	P2	4 years	P2T1	6
			P2T2	6

3.5 Data Collection

The data collection techniques of this research are structured by the levels of data collection techniques by Lethbridge et al. (2005). In their research on data collection techniques for software engineering research, three levels of data collection techniques are defined as the *first*, *second*, and *third degrees* [106]. Having data collection methods from the three levels provides the advantage of gaining insights from different perspectives for this qualitative research. The first-degree techniques involve direct involvement with the researcher and the subjects, while data is collected simultaneously [106]. Interviews are a first-degree technique, which involves at least one researcher asking questions to at least one respondent [106]. For this research, we adopt semi-structured interviews as it provides the ability to gain information from the subjects according to the defined script as well as a follow-up on open questions. We conduct two types of interviews. Type 1 interviews are aimed at getting to understand the project, the chosen agile approach, RE activities, and the documentation used in the project. Type 2 interviews are targeted at understanding the current state of documentation in each team with regard to the variables outlined

in Figures 3.4 and 3.5.

Furthermore, the second-degree data collection techniques define techniques that do not require direct contact between the subjects and the research [106]. Here, we combine the direct observation approach with experimentation to validate the proposed framework with two multi-team software projects. With passive participation, we gain insights into how effective is the use of the proposed framework within each team. Subsequently, the third-degree of data collection techniques "attempt to uncover information about how software engineers work by looking at their output and by-products" [106, p. 329]. With this technique, we adopt documentation analysis. We look at the documentation produced and used by the team (output) and the tools used in this process (by-products). Table 3.2 shows how these three selected data collection techniques are used to answer each sub-question.

Table 3.2: An overview of research methods per sub-question

sub-research question	**Data collection methods**
SRQ1: How is Agile Software Development (ASD) adopted within a multi-team software project?	Semi-structured interviews.
SRQ2: Which requirements engineering practices can be identified in the adopted ASD approach?	Semi-structured interviews.
SRQ3: Which kinds of documentation artefacts are used for the various requirements engineering practices within the ASD approach?	Semi-structured interviews and documentation artefacts analysis.
SRQ4: What is the current state of the identified documentation (from SRQ3) within the project?	Semi-structured interviews and documentation artefacts analysis.
SRQ5: What is the impact of requirements change on the identified documentation (from SRQ3) within the project?	Semi-structured interviews.
SRQ6: What framework can be used to address documentation issues by improving documentation practices in multi-team software projects?	Semi-systematic literature study and applying the guidelines of the treatment design by Wieringa (2014).
SRQ7: What is the effectiveness of the proposed framework from SRQ6?	An experiment with observations and a validation survey.

3.5.1 Data Collection Procedure

Before the start of the case study, consent forms are sent out to the participants to inform them of the objective of the research and assure that the collected data will be treated confidentially. The consent form can be found in Appendix B. The study of each case begins with a preliminary study (we refer to this as Type 1 interviews). A semi-structured interview, with a duration of at most one hour, is conducted with the Team Manager or Scrum Master of each project, to gain insights into the project and understand how ASD is embedded in the way of working. The questions for this type of interview can be found in Appendix C. The interviews are recorded with the permission of the interviewee and are transcribed. Additionally, documentation analysis is conducted to gain insights into the kinds of documentation used by the project and gain a general idea of the state of documentation. A combination of these two techniques answers the

first sub-research questions.

The next step after the preliminary study is to conduct semi-structured interviews (Type 2 interviews) in accordance with the goal and metrics defined in Section 3.1 to answer the fourth and fifth sub-research questions. The questions for this type of interview can be found in Appendix D. Documentation analyses involve making an inventory of the documentation relating to RE activities per category, comparing the documentation artefacts to analyse their quality based on a set of guidelines, and also studying what tools are used by the agile team to write documentation. Then, the problems identified with documentation are used as input for the literature study to design and propose a suitable treatment.

As discussed in with the GQM models in Section 3.1, the findings of the first four sub-questions are used as requirements input for the designing of a suitable framework using the techniques of design science treatment design. The treatment design phase of design science ensures that requirements are specified to aid the search for useful possible treatments [21]. Finally, the proposed framework is validated industrially using the treatment validation phase of the design cycle by studying the framework in an experiment to observe its effectiveness.

3.5.2 Pilot Study

According to Yin (2018), a pilot study is conducted as an initial phase of the data collection procedure to adapt the data collection procedure with respect to "both the content of the data and the procedures to be followed" [22, p. 106]. One of the potential cases for this research is selected for the pilot study due to the availability of participants in relation to the timeline of the pilot study. The interviews for the pilot study were scheduled to start in March 2022. For this research, the pilot study is conducted on a project with two agile teams developing a software package for a client. The teams are divided on a functional level. The insights from the pilot study are used to refine the case study protocol and is retained in the findings of this research. An overview of this case can be found in Section 4.1.1. During the data collection, we reach a point of theoretical saturation whereby the fifth interview was confirming information already collected in the previous interviews. Therefore, we decided to interview three team members in the subsequent case study. This also made it easier for C2 to commit to participating in the research as it entailed a lesser time effort.

3.6 Data Analysis

In this section, we present the various approaches to data analyses for answering the research questions. The data analysis triangulates data from semi-structured interviews, documentation analysis, and a treatment validation experiment. During the data analysis phase, the transcripts of the Type 1 interviews are analysed for problem orientation and familiarity with the adoption of ASD within the organisation and the project in the study. The findings from these interviews are used to guide the semi-structured interview questions of Type 2 interviews as well as the documentation analyses.

3.6.1 Semi-structured interviews

The transcripts from each of the interviews are analysed using thematic analysis. Thematic analysis is a data analysis approach used in multiple qualitative case studies, where recurring themes and issues are identified, interpreted, and conclusions are drawn from the findings [107]. First, the data is made anonymous and stored in the database of this research. For each case, the project is described as well as the agile teams. Subsequently, notes are taken by reading through each transcript to form initial codes. Patterns and differences across cases relating to the adoption of agile methods, RE activities, documentation processes and artefacts are noted in other to establish thematic patterns. For instance, the findings of RE activities in SRQ2 will be mapped to the phases of the RE cycle as shown in Figure 3.2. For SRQ3, an inventory will be made of which RE-related documentation is evident in the project, what are the rationales for use (see Figure 2.22), and how the use of tools influences the kind of documentation in the project. The data is visualised using a relational model. In Figure 3.9, we present a conceptual model of the potential flow of data in this research.

The flow of data in the conceptual model (Figure 3.9) begins with a project that executes an agile method, a combination of agile methods or an internally created method to scale agile to the project. A project consists of two or more agile teams, where each team consists of roles, processes, and tools. At least one role performs a certain RE activity as part of a process in the agile method and he or she may use a tool for that activity. An RE activity is an overarching term for elicitation, analysis, specification, validation, and requirements management-related activities. These types of RE activities are based on the phases of RE by Dieste (2009). Then, an RE activity may produce or use some documentation artefacts, whereby at least one role in the team is responsible for that artefact.

A documentation artefact is shaped by the rationale of use by the agile team, tool, and agile method. The type of a documentation artefact can be classified as either a product, process, core agile method or non-agile method artefact, based on the findings of Wagenaar et al. (2015). Moreover, the quality of a documentation artefact is assessed by quality categories based on the categories defined by Aghajani et al. (2019). This quality category is an input for the treatment requirements, which contributes to the design of the treatment. The satisfaction of the requirements towards the designed treatment is reached if the treatment is suitable for use in practice. In turn, the designed treatment is validated by measuring its effectiveness with regard to the variables defined in the research of Green et al. (2005). Table 3.3 presents an overview of the semi-structured interviews conducted in this research.

Table 3.3: An overview of the conducted interviews.

ID	Role of interviewee	Team ID	Date and time	Format
1*	Team Manger	P1T1	18th March 2022, at 13:00	Online
2	Architect/Lead Developer	P1T1	20th April 2022, at 15:30	On-site
3	Analyst	P1T1	22nd April 2022, at 10:30	On-site
4	Analyst	P1T2	29th April 2022, at 10:45	Online
5	Architect	P1T2	10th May 2022, at 13:30	On-site

Table 3.3: (continued from previous page)

ID	Role of interviewee	Team ID	Date and time	Format
6*	Scrum Master	P2	16th May 2022, at 15:00	On-site
7	Team Lead/Lead Developer	P2T1	30th May 2022, at 13:00	On-site
8	Product Owner/Domain Expert	P2T2	30th May 2022, at 14:30	Online

* means the interview is a Type 1 interview, whereas the others are Type 2 interviews

Additionally, in Table 3.4, we define the themes for each category using the sub-categories and aspects from Figures 2.17, 2.18, and 2.19 as inspiration.

Table 3.4: An overview of the themes defined for the data analyses.

ID	Theme
1	**Information content (What)**
1a	**Correctness**
1a1	Preciseness of documentation
1a2	Adherence of documentation to the given templates
1a3	Examples of the most problematic documentation to keep correct
1b	**Completeness**
1b1	The extent to which requirements are documented
1b2	The extent to which design is documented
1b3	Accuracy and completeness of references used in user stories
1c	**Up-to-dateness**
1c1	The consistency of the documentation with the working software
1c2	Examples of functionality in the code that is not yet documented
1c3	The extent documentation is sufficient to extend or maintain the application
1c4	The extent to which the translation in the documentation is outdated
1c5	The challenges faced when ensuring documentation is kept up-to-date
1c6	The difficulty in tracing the versioning and updates of the documentation
2	**Information content (How)**
2a	**Maintainability**
2a1	The difficulty in adding changes to the requirements documentation
2a2	The difficulty in adding changes to the design documentation
2a3	The awareness of how a change in one documentation impacts the others
2a4	The challenges of incorporating change requests in the documentation
2a5	Addressing the question of what is enough documentation
2b	**Readability**
2b1	Processes in place to review documentation and ensure that they are clear to read
2b2	The conciseness of the contents of the documentation
2c	**Usability**
2c1	Issues faced when using the various tooling to write documentation
2c2	The level of difficulty for external stakeholders to find information in the various documentation
2c3	Processes in place to ensure shared documentation is maintained correctly by both teams

Table 3.4: (continued from previous page)

ID	Theme
2d	**Usefulness**
2d1	Processes in place to encourage the various stakeholders to review and give feedback on the documentation
2d2	Planning and incorporating feedback in the documentation process
3	**Process Related**
3a	Internationalisation
3b	Contribution to documentation
3c	Automated documentation
3d	Other issues
3e	Strengths of documentation processes

3.6.2 Documentation analyses

Documentation analysis is performed on the documentation artefacts relating to RE activities to provide insights on the types of documentation used and the quality of that documentation. The quality of documentation cannot be assessed based on the documentation analysis alone, therefore the Type 2 interviews are used to support this objective.

3.6.3 Treatment validation experiment

The purpose of this experiment is to provide an understanding of how the proposed framework is adopted in practice and how effective is it in improving documentation practices. The treatment is applied to two multi-team software projects. The notes taken from the passive observations will be used to support the analysis of SRQ7. Similar to the semi-structured interviews, the thematic analysis approach is employed to derive codes related to the variables outlined in Figure 3.7. To conclude the experiment, a survey is given to all the participants to give their feedback on the use of the proposed framework. These insights enrich the findings from the observation. A detailed description of the experiment is provided in Chapter 7.

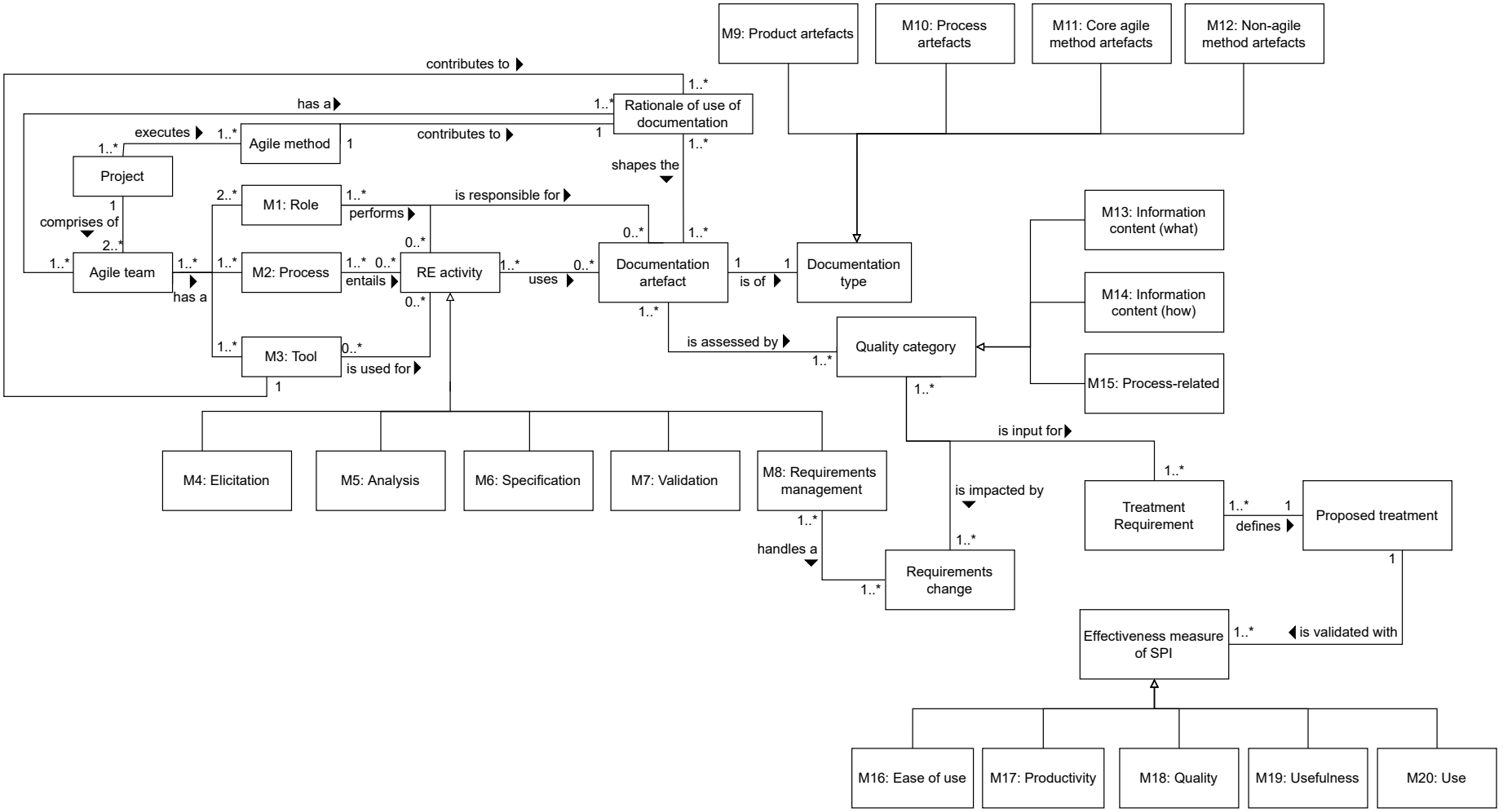


Figure 3.9: An overview of the conceptual model of the empirical data in this research.

Chapter 4

Case Descriptions

We present the first set of results from the multi-case study conducted as part of the Problem Investigation phase. We introduce the explored cases in Section 4.1, then we discuss the agile approach and evidence of RE activities in Sections 4.2 and 4.3, respectively. Finally, we present an overview of the findings of the use of documentation artefacts within the studied projects in Section 4.4.

4.1 Introduction to the Cases

The case studies involved interviews and documentation analysis of two projects within two different companies. Due to confidentiality and privacy reasons, we refer to these cases as P1 and P2.

4.1.1 P1: Education

P1 is a project within C1, that builds a solution for the educational sector. The solution consists of two web applications for the same client. Both applications concern the resource planning of staff in the educational sector. The project consists of two teams (P1T1 and P1T2), each web application is developed by one team. This implies that the teams are divided based on functionality with a common goal to align and reuse components. The development of the products within P1 has a long-term duration and has been ongoing for two and half years (as at the time of interviews).

P1T1 is currently developing a web application as part of the solution for a client, to assist the reintegration of teaching staff into the workforce after a long-term absence due to illness or disabilities, for instance. The team is responsible for developing and maintaining the frontend of the application, whereas a third-party provider is responsible for the development of the backend. The team consists of eight members at the time of interviews. This number is subjected to change as the up-scaling and down-scaling of teams is an ongoing activity for projects in C1. Initially, the team consisted of fourteen members in the [DevOps] phase, and this number was down-scaled to eight. The project is currently in a 'sort of a handover phase' (Team Manager, Interview 1), where

the functional delivery has been completed for the first release and it is in the process of transfer to the maintenance team while the team still develops new functionalities in parallel. The team manager describes this process as such. The first releases 'have gone to production already, so it is a hybrid situation where the project team is working on new features and the maintenance team will pick up the production incidents' (Team Manager, Interview 1). P1T1 delivers increments of the product to the production environment monthly with a process known as the monthly release.

P1T2 also develops a web-based solution for resource planning for educational institutions. What differentiates their product from P1T1 is that, the projects divide the teams based on functionality. Different functionality and different target users are a core distinguishing feature of the end-products of both teams. P1T2 consists of 6 members at the time of the interviews. This number, like that of P1T1, is subjected to change as up-scaling and down-scaling is an ongoing activity for projects in C1. In contrast with P1T1, where the team deploys a monthly release, the releases in P1T2 are less frequent. P2T2 deploys their product increment on a quarterly basis to the production environment.

4.1.2 P2: Civil Engineering

The second project explored is P2. P2 is a 'long-term' project aimed at optimising the processes and work of the design engineers within C2. The project consists of four solutions which are developed by two teams within the project, for confidentiality, these teams are referred to as P2T1 and P2T2. The solutions are used to visualise construction structures as well as to make various calculations for these structures. Currently, the solution is used internally with the aim of extending it to external users. 'Our main stakeholders will be the users of the software, and [..] the project manager is involved in the milestones of the project' (Scrum Master, Interview 6).

The teams are divided based on functionality, but also different infrastructure is used among the teams to deliver the functionality. P2T2 mainly consists of Digital Engineers who are writing scripts to perform calculations for bridges. 'So, they are delivering tools, and the users can use the tools to make the decisions for the bridges. But they are mainly involved in calculating the bridges' (Scrum Master, Interview 6), whereas P2T1 entails .NET and Unity developers developing an application to validate the calculations. Additionally, there is one Product Owner for each team and a Scrum Master responsible for both teams.

4.2 Agile method

We identified two different approaches to large-scale agile software development. In P1, an internally created method by C1 is used. This method is referred to as C1M in this research, due to confidentiality. On the other hand, P2 adopts SAFe. The methodology adopted by each project reflects the roles, processes, and tools used by the various projects.

4.2.1 Roles

Across cases, there are some similar roles. In Table 4.1, we present a role matrix highlighting the similar and different roles across the projects. Then, we explain what each role means.

Table 4.1: A matrix of roles identified within the teams of P1 and P2.

Role	**P1T1**	**P1T2**	**P2T1**	**P2T2**
Scrum Master	o	o	✓	✓
Analyst	✓	✓	x	x
Architect	✓	✓	e	e
Tester	✓	✓	✓	x
Developers	✓	✓	✓	x
Lead Developer	o	o	✓	x
Product Owner	e	e	✓	✓
Digital Engineers	x	x	x	✓
Domain Experts	x	x	✓	✓

o = Role exists with another name, ✓= Role exists with exactly the same name, e = External role outside the team, x = Role does not exist.

- *Scrum Master.* Both teams in P2 share the same Scrum Master. Within each team, the Scrum Master is responsible for coordinating the activities of the team as well as moderating the Scrum events of the team. Similarly, in P1, the role of the Scrum Master is adopted by the team manager. The team manager performs a similar function to the role of the Scrum Master in P2, but with additional project management duties. Per team in P1, there is a different team manager, who ensures 'that all the team members are happy and, working on the right [requirements] towards the customer, [..] managing customer relationships, tracking status, and communicating it back [to the customer]' (Team Manager, Interview 1).
- *Analyst.* The role of the analyst is evident in both teams in P1. They are either referred to as Analysts or Business Consultants within the company. In P1, an Analyst is responsible for gathering the functional requirements, organising workshops, and responsible for the analysis documentation. Also, non-functional requirements are handled using workshops, but the developers and testers ensure that it has been implemented correctly per user story. In P2 the role of the analyst is similar to that of the Product Owener.
- *Architect.* There is a shared Architect for both P1T1 and P1T2. The Architect guides the technical implementation of the project and advises on high-level decisions. For instance, they advise the customer 'on the right solution, [as well as] on [how Azure] components should be used [and] even which Azure services should be used and [..] on the security details' (Architect, Interview 2). For both teams in P2, the architect is not part of the team but is within the department, and he or she is consulted for help when there are questions from the team.
- *Tester.* The role of the Tester is similar across P1T1, P1T2, and P2T1. The Tester is responsible for 'perform[ing] all the manual and regression testing and making sure that we are delivering quality' (Team Manager, Interview 1).
- *Developers.* Both teams in P1 consist of frontend, backend, and full-stack Developers. Whereas, P2T1 consists of application developers in .NET and Unity. The Developers are responsible for implementing the user stories planned in the sprint.

- *Lead Developer.* The role of a Lead Developer is similar for teams P1T1, P1T2, and P2T1. 'That is the person responsible for the technical implementation' (Team Manager, Interview 1). As well as, coaching the developers and helping them when there are any questions. Also, in P1, or more generally in C1, Lead Developers have the job title of a Senior Specialist.
- *Product Owner.* In the teams of P2, the Product Owner is responsible for maintaining the requirements documentation such as the feature descriptions and user stories. And they 'will have some insights from the domain experts for it' (Scrum Master, Interview 6).
- *Digital Engineers.* Even though Digital Engineers perform a similar role as Developers in an agile team. They are referred to as Digital Engineers in P2T2 because they 'are writing scripts [.. and] they are delivering tools and the users can use [..] to make the decision for the bridges. But they are mainly involved in calculating the bridges' (Product Owner, Interview 8). In other words, they are also using the solution they develop in their daily work and they use tools like Grasshopper[1] to develop their scripts as opposed to the developers who use Visual Studio as a development environment.
- *Domain Experts.* The Domain Experts in both teams in P2 provide explanations of the domain knowledge in the field of civil engineering for various features and requirements within the project.

4.2.2 Processes

We discuss the activities of the teams toward internal and external communication as an essential aspect of agile methods. Also, we present C1M, the internally created method used by P1 and the adoption of SAFe by P2.

Internal and external communication

A fundamental aspect of ASD methods is communication and adaptability to change. In this part of the section, we introduce the approach of the explored cases to internal communication (within the team) and external communication (across the teams and with other stakeholders).

Both teams in P1 are collocated in the same office and are mostly seated together on one floor on the company's premises. In addition to working on-site, there is also a hybrid situation where the team members work from home occasionally. This is similar to that of P2, where the teams are also located in the same office on C2's premises. However, the teams mostly work from home. 'most of the time we work from home, [P2T1] is mostly at the office but [P2T2] is mostly working from home, so I should say they are not [always located] in the same building' (Scrum Master, Interview 6). Within each team, there is a Daily Scrum held every morning to discuss the progress of the user stories, tasks and any impediments. P1 and P2 use Azure DevOps to keep track of the requirements and manage the backlog. Communication within each team or across teams is also done via Microsoft Teams in the form of messaging and calls. But when there are complex topics to discuss, P2 adopts face-to-face meetings at the office, even though they mostly work from home. Also in both

[1] https://www.rhino3d.com/6/new/grasshopper/

projects, 'some team members are involved in [..] both teams' (Scrum Master, Interview 6), as highlighted in Section 4.2.1.

Although both teams in P1 are building separate applications as part of one solution, there is not much alignment needed in terms of functionality, rather, reusability of components is required by the customer. 'The customer requires that we do not do double work, so the alignment that is needed is mostly done in chapter meetings[2]' (Team Manager, Interview 1). 'We have these chapters [for] the frontend chapter and backend chapter. [..] We [hold it] weekly and the idea behind this is for knowledge sharing. Otherwise, there are small islands within the teams in the [P1] team. And usually, we can share code between the projects, so that is one reason to have the meetings' (Architect, Interview 2). Per team, there is someone responsible for the frontend or backend chapter meetings to align on a high level what the team is working on and whether resources can be shared between the teams. There is the same architect in both teams to ensure that the high-level architectural aspects are aligned. Moreover, for the alignment of design and analysis, the analysts from both teams have a weekly meeting to discuss components that may be reused among the teams. Also, a team lead stand-up is held to discuss resourcing within the teams when necessary.

Similarly, alignment is needed between both teams in P2 based on functionality. The scope of the project is such that functionality depends on each other, and therefore, the teams may depend on the functionality of each other from time to time. In contrast to the teams in P1, P2 does not share code across teams. This is mainly due to the different technology stack used by each team. Additionally, there is one Scrum Master for both teams, who ensures that the processes of both teams are aligned, and that dependencies are taken into account during the PI planning. P2 has a 'PI planning and a feature board, [as well as] dependencies between the two projects which are doable so that is where we align. And if someone has moved something [in the] planning, then we can adjust development as well' (Scrum Master, Interview 6).

Moreover, members in P1 make use of the Centre of Excellence (CoE) across the global company (C1). The CoE is a platform where employees within the branches of C1 in The Netherlands and other countries can also contribute to knowledge sharing and help each other. 'There is the Centre of Excellence [..], so you can ask questions, and maybe there are [also] best practices or examples out there' (Architect, Interview 2).

Furthermore, since the solution of P1 is developed for external use. There are few media of communication between both teams and the customer. There is a weekly status update meeting held with the customer and the scrum masters of both teams. 'During this status update is where officially both teams explain what the status is, what the progress and impediments [. . .], and such. Also, [this is] where resourcing can be discussed' (Team Manager, Interview 1). Additionally, team members can have direct contact with the client via email or verbal communication. 'However, the general rule is that if something has been decided or if questions are bigger than just a simple yes or no to proceed, then it should be monitored through the SharePoint' (Team Manager, Interview 1).

[2]A chapter meeting is used to share knowledge among the developers per team.

The SharePoint environment is a customised platform used by both P1T1 and P1T2. This is also discussed in Section 4.2.3.

The following alignment meetings are used to align the processes between the teams and/or external stakeholders.

- *Bi- or tri-weekly architecture meetings (P1).* This meeting is held with lead developers and architects from both teams to align the high-level architecture and reusable components.
- *Frontend and backend chapter meetings (P1).* A weekly meeting is used to share knowledge among the developers in both teams. 'We have these chapters [for] the frontend chapter and backend chapter. [..] We [hold it] weekly and the idea behind this is for knowledge sharing. Otherwise, there are small islands within the teams in the [P1] team. And usually, we can share code between the projects, so that is one reason to have the meetings' (Architect, Interview 2).
- *Technical stand-up (P1).* This meeting is held three times per week to discuss technical challenges and solutions within the team. It is held on 'Monday, Wednesday, [and] Friday, [.. where] developers are involved plus architects,' (Architect, Interview 2).
- *Weekly design and analysis meeting (P1).* 'There is also a weekly design and analysis meeting. And also yeah, it's used to align the design between the two teams' (Analyst, Interview 3).
- *Weekly status update meeting (P1).* 'The progress is monitored in a weekly session with the customer. So, during this status update is where officially both teams explain what the status is, what progress and impediments [are], and such. Also, where resourcing can be discussed' (Team Manager, Interview 1).
- *Agile Release Train (ART) Sync meeting (P2).* A biweekly meeting 'with the product manager, product owners and the scrum master. So mostly for this ART. So it's like an ART sync meeting [for] not only this project, but for all products in this ART' (Scrum Master, Interview 6). 'The ART sync is the common base for aligning the two teams' (Scrum Master, Interview 6).
- *Weekly PO Sync meeting (P2).* A weekly meeting with the product owner of each team and the technical manager to align the status of the projects with the long-term goals of the department.
- *Tri-weekly Guild meetings (P2T1).* These are meetings on a particular area of expertise to share knowledge across teams within the company. 'For example, we have a DevOps Guild and there, we discuss different DevOps topics because DevOps is not associated [with] one particular topic or one particular team. There are people from all the different teams. And also, [there is a] testing guild and cloud infrastructure guild' (Lead Developer, Interview 7).
- *Team alignment meeting (P2T2).* Within P2T2, there is a meeting that is organised occasionally to discuss their appointments and align the setup and guidelines of the code.

P1: Internally created method

Company C1 designed and uses an internally created method, we call it C1M due to confidentiality. C1M is based on an approach that combines aspects of Waterfall with agile software development. This method is based on agile with control. Working with fixed contracts, the aspects of time, budget, and quality

are fixed within the project. Therefore, changes to the scope of the requirements are handled by utilising change requests. 'In a truly agile sense, you cannot have all three aspects locked: time, budget, and quality. And for us, it is important that we make sure that the budget and the deadlines are set to an expectation based on what we agreed upon. So, this is then where it is really important to be clear about what you are supposed to build or guide the customer in change requests that might affect the deadline and maybe work towards an alternative deadline for changes so you can stick to the original plan as much as possible' (Team Manager, Interview 1).

C1M guides the projects to initiate with a clarification phase. This is the phase where requirements are analysed from the contract and existing documentation. Then the functional requirements are written as user stories and divided into workshops. Preparation for these workshops is done by preparing the user stories and prototypes, as well as determining the potential risks and problems. 'In these workshops, we gather all necessary stakeholders from the customer from third party providers and we go through all the user stories which are not clear to us' (Team Manager, Interview 1).

Subsequently, a core principle of C1M is to ensure that decisions are recorded as much as possible and approved by the client. The team manager of P1T1 explained the core principle of the company's methodology as 'the process of recording decisions and actions in a set state or in a set of predefined documents which allow you to be very explicit in what you are doing and what you will do' (Team Manager, Interview 1). Documentation of decisions and the approval of the client is very crucial to the C1M. The company's SharePoint environment, provides extensive guidelines and templates for documentation at every stage of the SDLC. The approval of documentation and decisions is also done via SharePoint. However, P1T1 deviates from the tooling aspect of the methodology. P1 uses Azure DevOps to coordinate development activities and manage the backlog. Therefore, the approval of user stories by the client is done via Azure DevOps instead of SharePoint.

After the backlog has been defined, the functional requirements in the backlog are planned into sprints. The development of the system in the teams is guided using Scrum with sprints of two weeks. The following Scrum events are utilised by each team using the DevOps phase.

- Sprint Planning
- Daily Scrum
- Sprint Retrospective (every two sprints for P1T1 and every sprint for P1T2)
- Refinement session
- Sprint Review

During the sprint, there is also a change management process that takes place to handle new requirements to the original scope of the project. These requirements are referred to as change requests and are handled by creating a backlog item known as a change. The change is then linked to a user story in Azure DevOps. For each team, there is a release at the end of each sprint which is deployed to the Functional Acceptance Test (FAT) environment for the client to

test and report defects. Besides the FAT testing, a monthly or quarterly release is deployed to the Production environment by P1T1 and P1T2 respectively. Additionally, the output of the retrospective may be process improvement actions referred to as cases which are added and tracked in SharePoint.

P2: SAFe

P2 adopts SAFe as introduced in Section 2.4.1. However, some adjustments are made to the method. Within the Digital Transformation department of C2, Agile Scrum is used by most teams. However, P2 adopted SAFe about half a year ago (at the date of Interview 6), to improve the collaboration among the teams within the project. The portfolio version is used as shown in Figure 2.10.

Moreover, a Program Backlog and a Kanban board per team in Azure DevOps are utilised to keep track of the progress. Since the experience with the use of SAFe within C2 is fairly new, not all the elements are used as yet. These are gradually being integrated into the project. The main reason why P2 adopted SAFe is explained by the Scrum Master. 'There was a need for more collaboration between the teams and we were looking for a framework and that which was supporting that. And SAFe [..] is used a lot in The Netherlands, so we picked that one and we choose the [SAFe activities] which are really helpful for us at the moment. And I have to say there are not a lot of multi-team projects yet in [C2]. But this is really the first one, and especially in the current PI we saw it, it was very helpful that [..] during the PI planning and we could [clarify] what are the dependencies between the teams' (Scrum Master, Interview 6).

The following activities are adopted by P2 at the time of interviews.

- *Continuous Deployment Principle.* 'For other projects, we [are] using that more and more, but especially for the product of [P2T1], it is still a standalone application. And for [P2T2] we deliver scripts so we still have to have to find out what is the best way to release the software' (Scrum Master, Interview 6). In P2T1, they have a website where the application is made available for download for the end-users. The application is deployed automatically, but for P2T2, they do not have automated deployment yet.
- *Design Thinking.* Design thinking is adopted within the project to some extent whereby feedback is collected from users and used to improve the solution. 'At least we start with [a] pilot bridge. So, we start finding out [whether] the software is usable [..] for the type of bridge. So, that could be a kind of design thinking. But as a team, we are a bit further away from the customer, [..] and for [the solution of P2T1], the user starts using it and find some things to improve' (Scrum Master, Interview 6).
- *Implementation Roadmap.* There is a roadmap for both teams which describes the 'planning for the project, [and the] milestones in it. [It] connects the rest of the year, so multiple PI's [are specified in the] roadmap' (Scrum Master, Interview 6). The teams have an overview of what is being developed. 'They know what is on the PI Planning, so they know what they are doing and. So, they are aware of the features they are making, so for some features, they are depending on the other team, and they are aware of it' (Scrum Master, Interview 6).

- *PI Planning Event.* A periodic event is used to plan the epic and features for the coming Product Increment (PI) as well as the dependencies among the teams.

During the PI planning phase, the Product Owner of each team organises the team backlog by planning and defining the features as well as the feature descriptions and user stories. After the team backlog has been defined, the development of the PI is planned into a bi-weekly sprint using a Sprint Planning meeting for each team. The team backlogs of P2T1 and P2T2 are therefore on separate boards in Azure DevOps, as the dependencies are already identified during the PI planning event. Similarly to the teams in P1, Scrum is used to guide the DevOps activities of each team. And the same Scrum events as P1 are used, and the Retrospective is conducted at the end of each sprint.

Using automated deployment P2T1 deploys after the completion of each user story. P2T2 on the other hand does not deploy as yet, but the scripts they develop are immediately executable and ready to be used by the Digital Engineers themselves or the other engineers within the company. Lastly, there is a 360 Degrees Feedback Session for each team in April and October, where team members give each other feedback on their performance.

Process Models

Using BPMN 2.0 we designed process models for the happy flow processes of the SDLC from requirements elicitation to deployment for all the teams. To understand the difference between the documentation artefacts, we provide a legend in Figure 4.1, that distinguishes formal documentation artefacts from informal documentation artefacts. In this context, the formal documentation artefacts are created using Microsoft Word and fixed templates prescribed by the agile methodology used by the project.

Figure 4.1: A legend showing the types of documentation artefacts used within the process models.

The processes are divided into two phases for each team. The first phase is the requirements elicitation phase whereby at least the high-level requirements are elicited. For both teams in P1, this is referred to as the clarification phase and in P2, this is referred to as the PI planning phase. The activities of the clarification phase for P1T1 and P1T2 are similar with some different sub-activities, whereas the same activities are used in the PI planning phase of both teams in P2, due to the guidelines of SAFe. Therefore, we present these phases of P1T2 and P2 in Figures 4.2, and 4.3, respectively. The activities during the DevOps phase of P1T1 and P1T2 are similar, therefore we illustrate an overview of these activities in Figure 4.4. Additionally, the process models for the activities during the DevOps phases of P2T1 and P2T2 can be found in Figures 4.5 and 4.6, respectively.

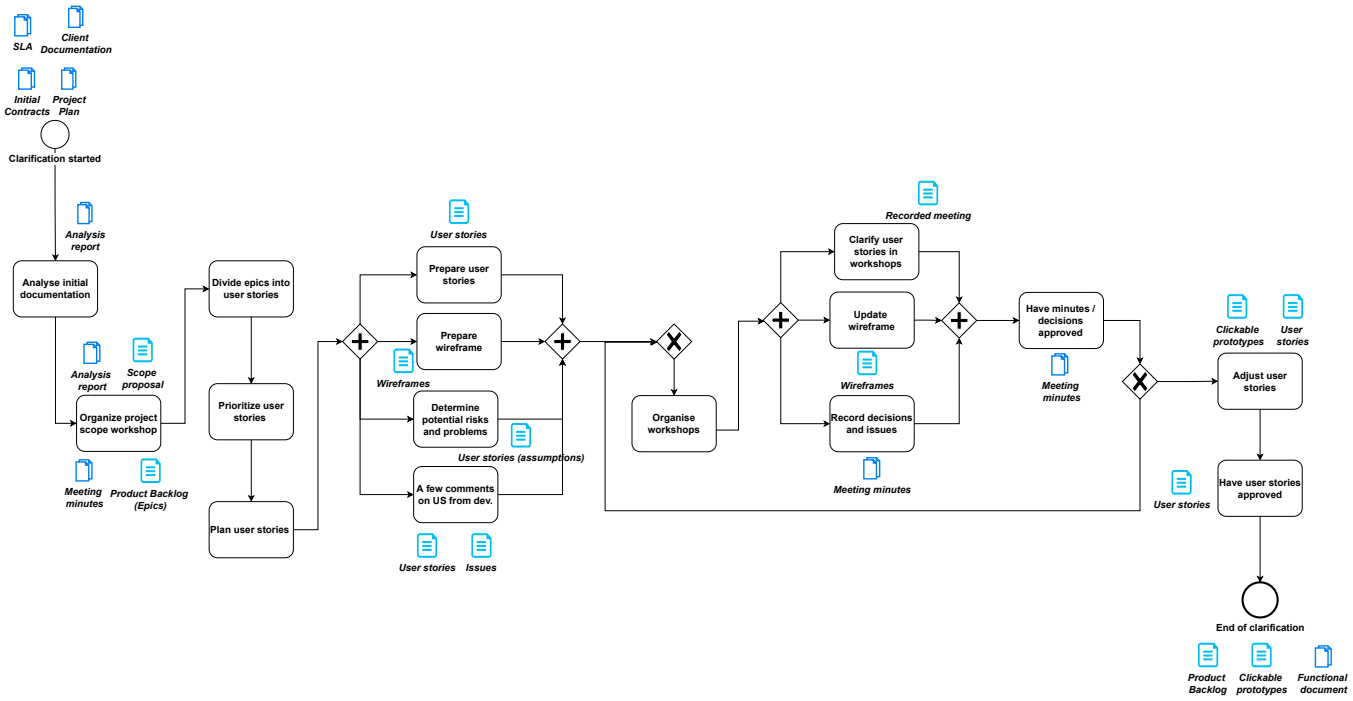


Figure 4.2: A process model showing the activities during the clarification phase of P1T2.

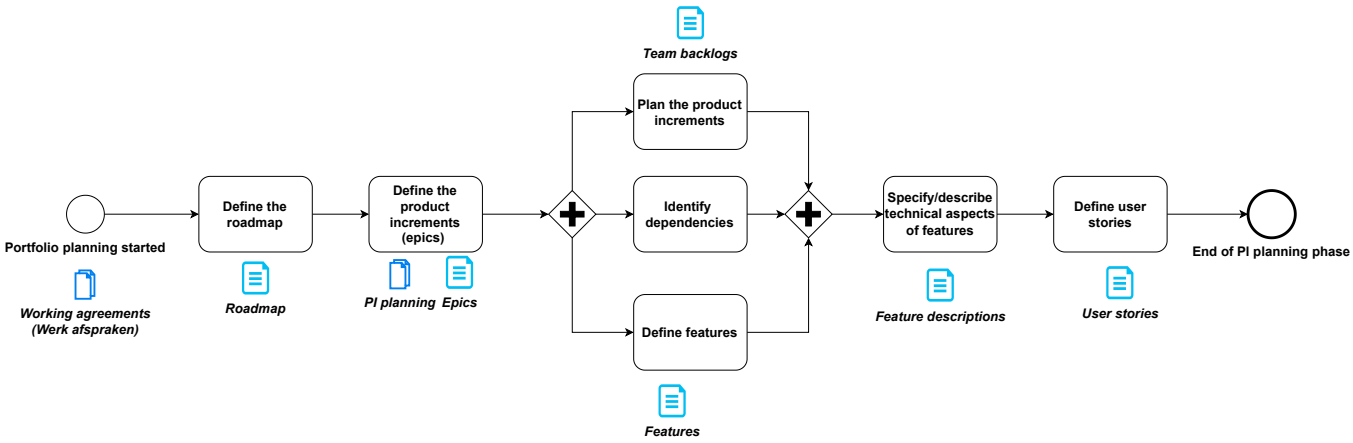


Figure 4.3: A process model showing the activities during the PI planning phase of P2T1 and P2T2.

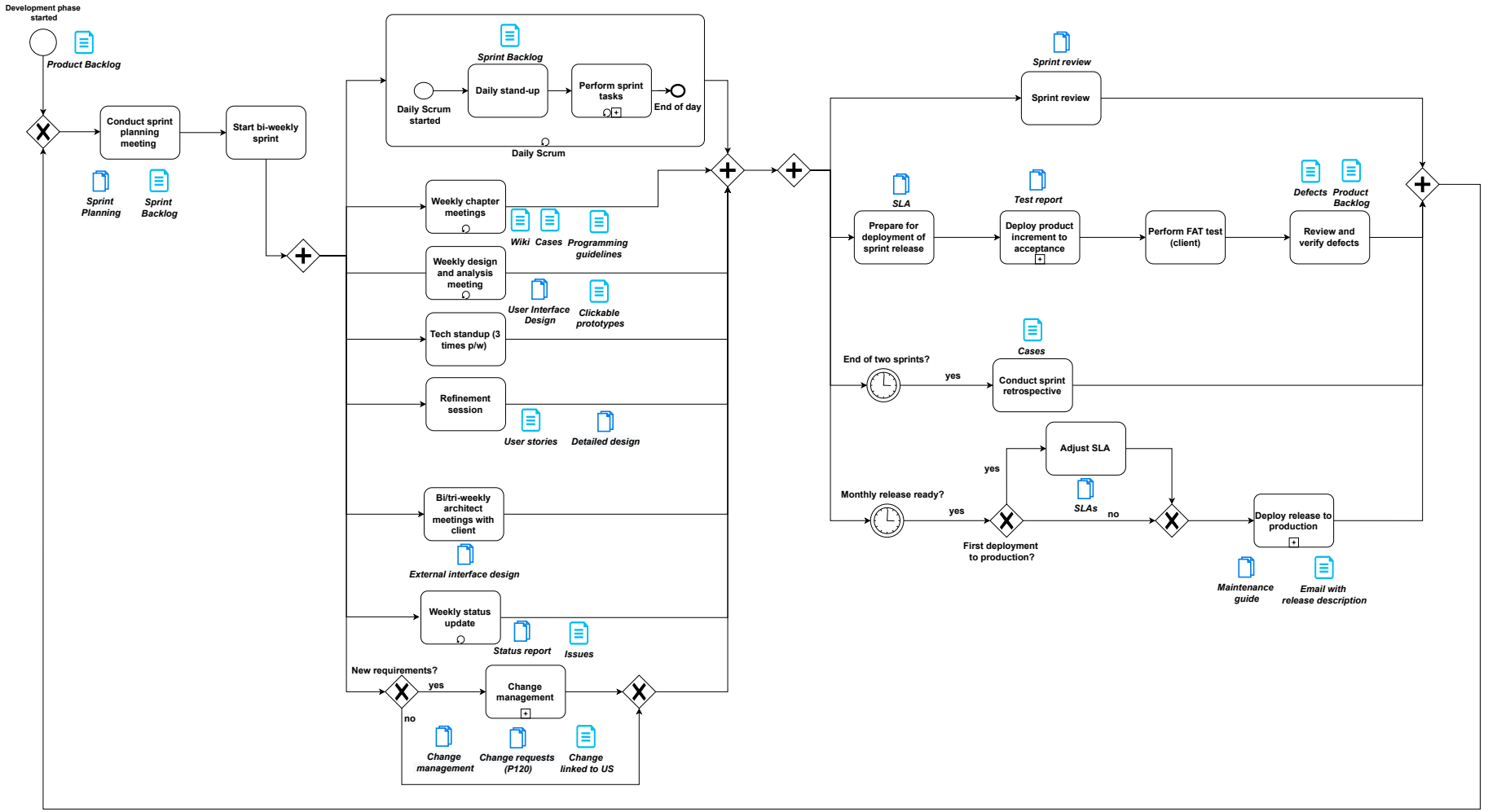


Figure 4.4: A process model showing the activities during the DevOps phase of P1T1 and P1T2.

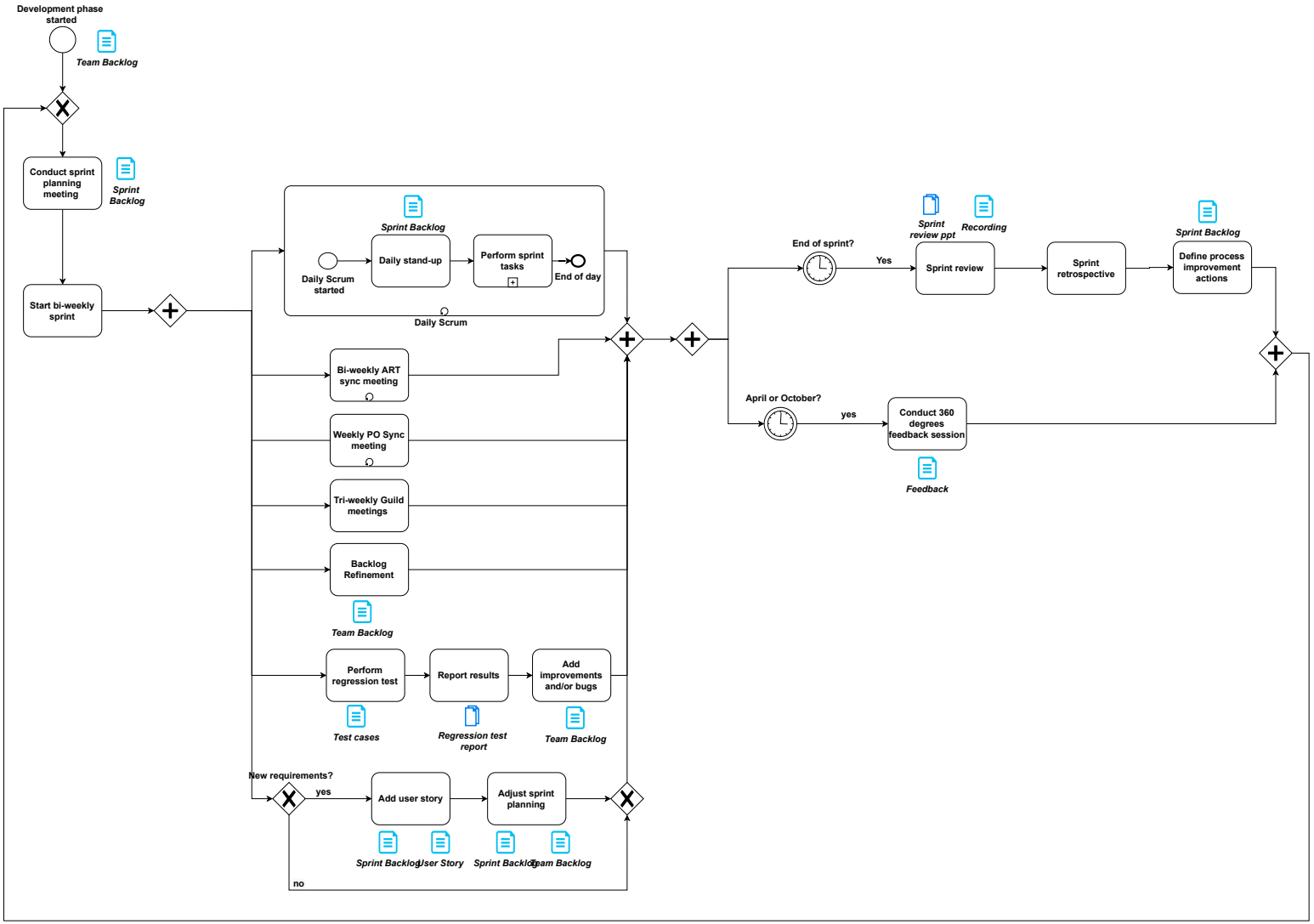


Figure 4.5: A process model showing the activities during the DevOps phase of P2T1.

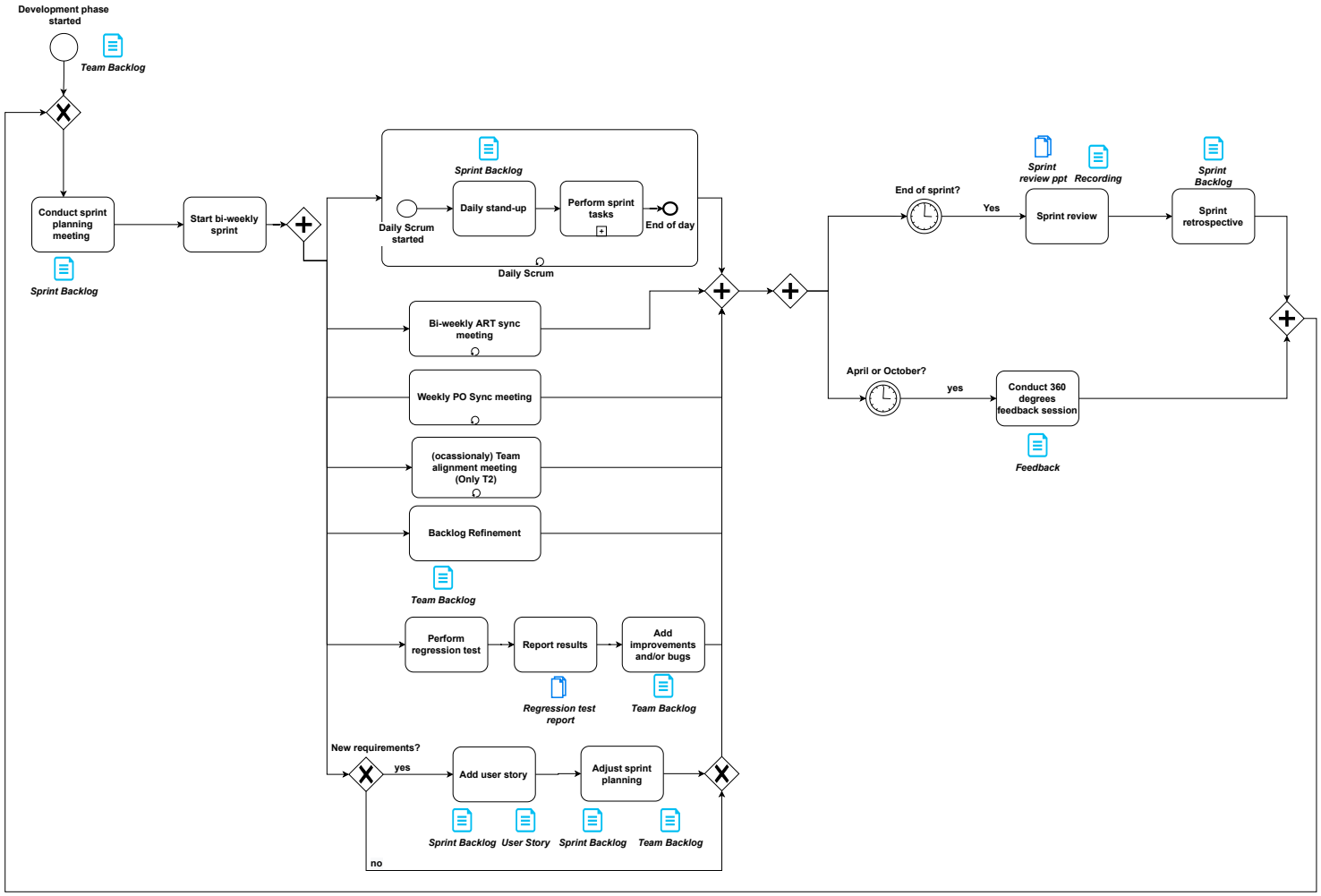


Figure 4.6: A process model showing the activities during the DevOps phase of P2T2.

In Figures 4.4, 4.5 and 4.6, we see the 'Perform sprint tasks' activity with sub-activities. The sub-activities for P1T1 are the same and it is shown in Figure 4.7. On the other hand, the sprint tasks differ per team within P2 due to the way of working of the teams and the fact that P2T2 develops scripts without a UI. These sub-activities are shown in Figures 4.8 and 4.9.

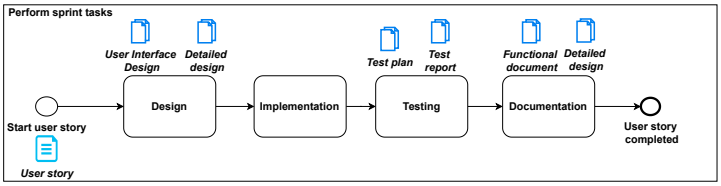


Figure 4.7: An overview of the sub-activities in 'Perform sprint tasks' in P1T1 and P1T2.

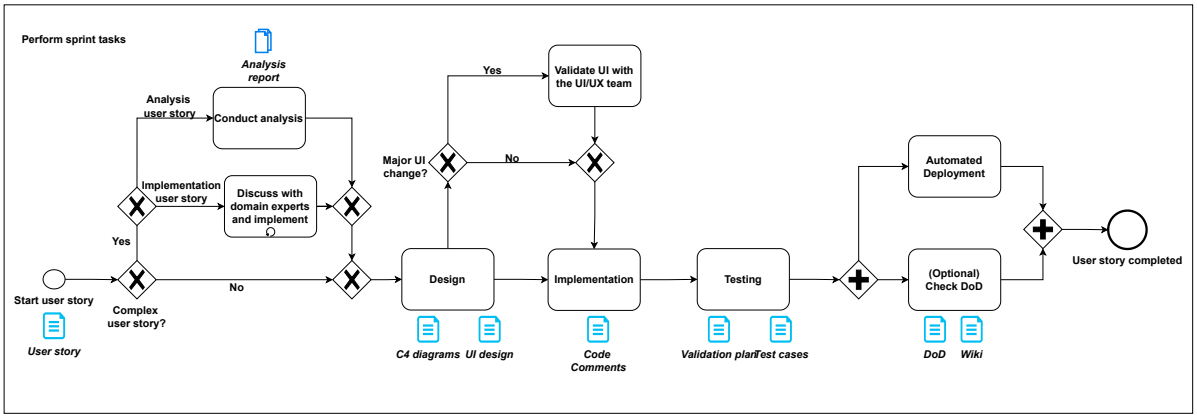


Figure 4.8: An overview of the sub-activities in 'Perform sprint tasks' in P2T1.

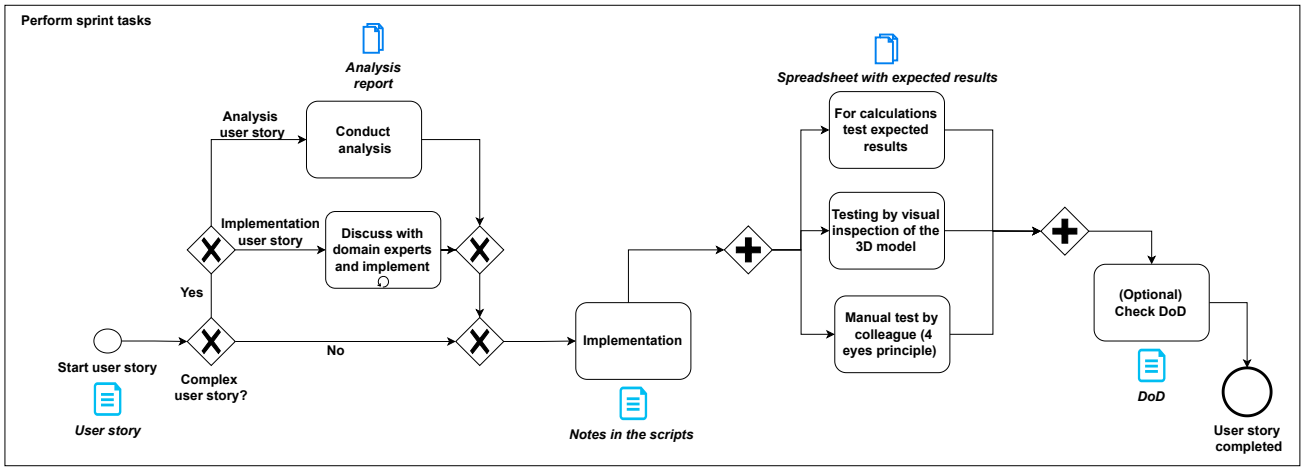


Figure 4.9: An overview of the sub-activities in 'Perform sprint tasks' in P2T2.

4.2.3 Tools

In Table 4.2, we present a tool matrix highlighting the similar and different tools used across the projects in writing documentation. Then, we discuss the purpose of each tool.

Table 4.2: A matrix of tools used for documentation identified within the teams of P1 and P2.

Tool	P1T1	P1T2	P2T1	P2T2
Axshare	✓	✓	x	x
Azure DevOps	✓	✓	✓	✓
DrawIO	✓	✓	x	x
Figma	✓	✓	x	x
PlantUML	x	x	✓	x
Microsoft Office	✓	✓	✓	✓
Microsoft Teams	✓	✓	✓	✓
SharePoint	✓	✓	x	x
Visual Studio	✓	✓	✓	✓
Wiki	✓	✓	✓	x

✓= tool is used by the team, - = tool is not used by the team.

- *Axshare.* Used to design wireframes to facilitate discussions at the workshop. The wireframes used by the teams in P1 are defined as such. 'We have wireframes [..] and it is unstyled. It is just black and white, just to define the structure' (Analyst, Interview 3). Based on this the validation of specific parts of a feature or user story is performed during the workshops.
- *Azure DevOps.* Used by both P1 and P2 to manage the backlog and keep track of the requirements. The functional requirements are written in the form of user stories. In P1, these user stories are also approved by the client in Azure DevOps.P2T1, also uses Azure DevOps to keep track of the test scripts for the manual tests conducted by the team.
- *Draw IO.* Used by the teams in P1 for modelling requirements by drawing UML diagrams such as sequence, flow, and use case diagrams.
- *Figma.* Used by the teams in P1 to mock out designs and prototypes which are then used as 'a clickable prototype in HTML' (Analyst, Interview 3).
- *PlantUML.* Used by P2T1 to generate C4 diagrams to document high-level architecture, by specifying 'which components you have and how those are connected, and then the program makes a layout for that automatically' (Lead Developer, Interview 7).
- *Microsoft Office.* Used by the teams to edit Word documents, design presentations for the Sprint Review, and email communication among others. The test cases for the scripts developed by P2T2 are also Excel files with calculations and expected results.
- *Microsoft Teams.* Microsoft Teams is used by team members to communicate with each other.
- *SharePoint.* A dedicated SharePoint environment designed and used by company C1, of which all the teams in P1 and the customer have access. 'All [formal] documentation is gathered in [SharePoint]' (Team Manager, Interview 1).

- *Visual Studio.* Used by all the teams. But it is used by P2T1 and P2T2 to write code comments. Code comments are an important documentation artefact in both teams to explain complex formulas in the code. As well as for tracking version history and changes in P2T2. In P2T1 a general rule whether to update the Wiki or code comments is, 'usually for small [notes or explanations] it is better to just use the code comments, but [when] it gets more complicated and it requires for example pictures, then something in the wiki is more appropriate' (Lead Developer, Interview 7).
- *Wiki.* Used by P2T1 to document the 'explanation of domain and [..] the working of the software' (Scrum Master, Interview 6). In comparison with P2T1, the teams in P1 do not use the wiki as often, they use it to keep track of 'a list of knowledge sharing topics' (Architect, Interview 2) to be discussed in the various chapter meetings.

4.3 RE Activities

In Section 2.3, we discussed the aspects of Requirements Engineering (RE). Here we present our findings with regards to the evidence of elicitation, analysis, specification, validation, and requirements management in scaled-agile software development projects.

4.3.1 Elicitation

The activities involved in eliciting requirements differ for the teams in P1 and P2. We discuss the approaches of both projects to requirements elicitation.

P1 develops a solution for an external client. Therefore, during the clarification phase of the C1M approach, the first source of information available to the teams is the initial contract documentation which includes the problem statement, an overview of the proposed solution by C1 as well as initial requirements with estimates for the fixed-priced project. By conducting *documentation analysis* from those sources, the analyst in each team identifies the initial requirements for the to-be system. Afterwards, user stories are defined at a high level due to the limited available information. The user stories are then planned into workshops. The aforementioned activities are a summary of the activities from '*Analyse initial documentation*' until '*Organise workshops*' in Figure 4.2.

Workshop is an elicitation technique used by both P1 to present and discuss the proposed user stories with the customers and occasionally, other external parties. During the preparation for a workshop, the assumptions of user stories are written, wireframes, and optionally technical designs are drawn. These are then clarified at the workshop with the client. The decisions are recorded as minutes in the Meeting Minutes artefact. Adjustments to the user stories may be made on the spot or afterwards by analysing the recorded decisions from the workshop. At the end of the clarification phase, the backlog is planned with user stories for the DevOps phase. During the DevOps phase, the product owner on the client side 'would provide [..] the description of the user story. This description will then be adjusted or refined, with the help of our analyst and possibly also developers' (Team Manager, Interview 1). And that forms the basis for new user stories or changes in existing requirements by means of *documentation analysis*. The clarification phase ends with a Product Backlog

of user stories ready to be refined. The process described entails the activities from '*Organise workshops*' until the 'End of clarification' in Figure 4.2.

On the other hand, P2 approaches elicitation differently since the project is guided by SAFe. As shown in Figure 4.3, each team makes use of the Working Agreements artefact, which specifies what kind of expertise is in the team and how the team works. During the PI planning phase, the Roadmap is defined by the Product Manager means of *discussions with the Civil Engineers and Domain Experts* within the department about the features to be developed for optimising the processes of the Civil Engineers within the company. The *Roadmap* entails an ordered list of functionality to be implemented in the coming years. The Product Manager defines 'the roadmap [as well as] *the epics*. And [the Product Owners] are allowed to make the features and the [user] stories. So [the] road map has mostly the combination of [..] the overview [and] the big picture and the product owner does the more specific details' (Product Owner, Interview 8). After the *features have been defined*, the Product Owner *adds the feature descriptions and decomposes the feature into user stories*. The user stories are then defined and ready to be refined by the agile team.

4.3.2 Analysis

Each team in P1 *models wireframes and technical designs* such as the External Interface Design artefact, before the workshops. These may be adjusted during workshops and afterwards, further details of the user story are written. Additionally, the functionality of the user story is *modelled as a clickable prototype*. This type of prototyping is used to model the flow of the application, what the inputs and outputs are, and what the User Interface (UI) looks like. During the DevOps phase, as shown in Figure 4.4, *refinement sessions* are used to discuss the user story, whereby the completeness and understanding of the US are checked. Then, the user story is updated to be more precise and clear. Additionally, user stories with a complex scope are also accompanied by requirements models such as UML diagrams using the Detailed Design artefact. 'in the case of a complex user story, [..] you would see more UML diagrams, mostly sequence or flow diagrams to describe how a certain flow is and use cases are used when [the functionality] is truly new' (Team Manager, Interview 1).

Moreover, one or more *design task(s)* form part of the tasks of a user story in P1, as shown in the process model of the sub-activities of a sprint task in Figure 4.7. A design task may either be additional UI designs which are documented in the User Interface Design artefact or a technical design such as a UML sequence diagram which is documented in the Detailed Design artefact. These tasks ensure that the user interface is clear as well as the technical flow of the user story.

Similar to P1, P2 uses of the *refinement sessions* to 'discuss the [user] stories and questions about it, and even estimate them right during the refinement in the [DevOps] phase. When they are estimated, and we know enough from the requested functionality and then we are ready to build' (Scrum Master, Interview 6). Another approach used by the teams in P2 is the use of a *Research user story* to handle complex user stories. 'It is depending on how complicated the user story is, but, normally [we] first [specify] kind of resource or research

user story, so we first analyse what the problem is, [..], write about the problem, and we present what kind of options we have to tackle the problem. And then we make a choice and show that could be kind of [..] the solution you want to implement, and the next year story will be to implement the proposed solution' (Scrum Master, Interview 6). If this user story to be implemented is not clear, then the developer may *discuss the complex details with the Domain Experts*. The analysis activities for a user story are similar for both teams in P2 as shown in Figures 4.8 and 4.9.

During the sprint, P2T2 does not have design activities as part of their user stories, this can be seen by comparing the sub activities of a sprint task in Figures 4.8 and 4.9. The reason for this is that P2T2 implements their user stories as scripts without a UI, and they do not design the architecture as well. On the other hand, P2T1 performs design activities similar to the teams in P1, but to a different extent. Only the *high-level architecture is documented*, as the team also uses code comments to explain specific reasoning and examples in the code. The high-level architecture is designed using C4 diagrams. For UI design, the decision is made whether the user story imposes a major change to the UI. *If the UI change is major, then it has to be validated with the UI/UX team* within the company.

4.3.3 Specification

For the teams in P1 and P2, requirements are specified as user stories in Azure DevOps using the hierarchy of *Epic*, *Features*, and *User Stories*. They are specified using the role, action, and benefit format. Also, for user stories in P1, they are expanded with acceptance criteria and a functional and technical description, as well as a link to the documentation. 'In Azure DevOps, all user stories have their descriptions and links towards all these documents' (Team Manager, Interview 1). Also, 'documentation is standardised in a template' (Team Manager, Interview 1), and the team 'usually refer to [the requirements] in the documentation or the other way around, [..] we say this requirement is covered by user story x' (Analyst, Interview 3). Documentation such as the User Interface Design artefact describing the UI, authorisation matrix, and client documentation is attached to the user stories of P1T2. Moreover, requirements in P1T2 are documented in three formats, user stories, requirements backlog, and business rules.

In P2T1 and P2T2, the *Domain Experts also attach documentation to the user stories* where necessary, to explain a certain topic or provide domain knowledge. '[As] a domain expert, I deliver documents' (Product Owner, Interview 8). These documents, in the case of P2T2, are attached as a Word document for instance, but not a link to a shared document.

4.3.4 Validation

User stories in both P1T1 and P1T2 are *reviewed* and *approved* by the customer, as shown with the '*Have minutes decisions approved activity*' in Figure 4.2. A core aspect of C1M is to record decisions and have them approved by the customer to ensure clarity and traceability. During the clarification phase, the Meeting Minutes and the User Stories are reviewed by the client and approved

before the Dev—Ops phase. Also, in the case of a new user story, the Product Owner provides a description that serves as the basis of the requirements for that user story. Subsequently, after a user story has been implemented, it is tested and reported upon using the Test Report. The activities of the testing process are illustrated in Figure 4.7. Afterwards, the Functional Documentation artefact is updated with the correct flow of the functionality. These are all documentation artefacts made available for internal and external stakeholders to validate requirements. After the product increment has been deployed, the *client tests and verifies the correct implementation of the user story* and reports any defects. Finally, there are *acceptance documents* to keep track of the status of each user story when it is delivered to the client. 'There is a set of user stories with an ID. Those will be listed as delivered on time and as expected, and that is something that you will then find in the acceptance documents' (Team Manager, Interview 1).

Both teams in P2 use the Refinement and Sprint Planning to clarify any questions or missing information in the user story. During the refinement, the Scrum Master assigns the user stories 'to the developers and plans them in [the current] sprint and or in the next sprint. And there are always extra questions or something, but normally we try to get most of it clear during the refinement and the planning is normally shorter meeting with a lot of assigning to people and priorities' (Scrum Master, Interview 6). In P2T1, *the user story is tested after the implementation*, these activities are illustrated in Figure 4.8. The tester writes the test scripts and tests the implemented user story according to those test cases. Afterwards, the Definition of Done (DoD), may be checked before the user story is closed. But that is not always the case. P2T2 takes a different approach to testing, there is no tester on the team and the test cases are not defined in Azure DevOps as P2T1. However, there are three different ways a script may be tested, these are shown in the latter part of Figure 4.9. If it is a calculation user story, it is tested using the attached Spreadsheet with the expected results. If the user story outputs a 3D model from the scripts, this is validated by visual inspection by another team member. Other user stories are tested by another team member ensuring the 4-eyes principle.

4.3.5 Requirements Management

Changes to the original scope or original requirements is inevitable in both projects. As mentioned earlier, in P1, there is a set scope based on a fixed contract. However, 'it is possible for the customer to introduce changes to the scope, which [is] then analysed and fit into upcoming sprints' (Team Manager, Interview 1). A change request process is used to handle new requirements within the sprint, which is shown in Figure 4.4. 'It is not like [the Analyst] would change an existing requirement in the document, but [she] they would rather *write a change request*' (Team Manager, Interview 1). Change requests are artefacts stored in the SharePoint known as changes, and a *change is linked to the user story*. But the original requirements from the clarification phase do not change, they are only expanded with more details. 'The requirements themselves will not change that much, only maybe more details' (Team Manager, Interview 1).

The process of handling requirements change in the teams in P2 is similar to

that of P1, however, different artefacts are used. In Figures 4.5 and 4.6, we illustrate the process of handling requirements change during the sprint. To handle a change, a new user story is introduced. 'You normally have a new user story, which is describing the change' (Scrum Master, Interview 6). Also, changing an existing user story depends on whether it is opened. 'It depends on if it's still open and we can change this and [it is] still in the Sprint. If it is closed, then we will not reopen it, and change it, [..]. Sometimes we use it as a kind of improvement and then it is like a small user story. But [..] this will be reflected in a new user story. And [.. in the] definition of done where it has a rule that we also check [whether] the wiki is still up to date if we change something. Because that is not automatically changed in the wiki. So, another thing is that we also look at the validation tests, [and] if something changes every Sprint, we do are regression test and there is a check. If there is new functionality, we change the regression test as well' (Scrum Master, Interview 6). The new user stories are also stored as user stories in Azure DevOps and if the DoD is used, it ensures that the Validation Tests and Regression Tests cases are updated.

4.4 Documentation Artefacts

We present the documentation artefacts used within the agile teams in P1 and P2. As introduced in Section 2.5.3, documentation can be categorised as a product, process, core agile method, or non-agile method artefact. In P1, the team also makes a distinction between technical and functional/analysis documentation. Technical documentation artefacts are mainly product documentation written by the developers and/or architects. During the case study, we identified from the Type 1 interviews an extensive list of documentation, especially in P1. Therefore, for the interview questions, we used groups of documentation relating to the purpose of the documentation such as requirements, design, and test-related documentation. There is a more exhaustive list of documentation artefacts used within P1, but we only discuss the documentation that is relevant to the clarification and DevOps phases. The use of documentation artefacts throughout the activities of each team can be understood from the process model in Section 4.2.2. In Tables 4.3 and 4.4, we present an inventory of documentation artefacts used by P2 and P1, respectively.

Table 4.3: An inventory of (but not limited to) documentation artefacts used by each team in P2.

Name of artefact (in English)	**Role responsible**	**RE activity**	**Type of artefact**	**Rationale of use**
Working Agreements	Scrum Master	-	PROC	Internal communication
Project planning	Product Owner	E	CAM	Agility
Program Backlog	Product Manager	S	CAM	Agility
Team Backlog	Product Owner	S	CAM	Agility
Sprint Backlog	Product Owner, Scrum Master	S	CAM	Agility
Epics	Product Manager	S	CAM	Agility

Table 4.3: (continued from previous page)

Name of artefact (in English)	**Role responsible**	**RE activity**	**Type of artefact**	**Rationale of use**
Feature Descriptions	Product Owner	S	CAM	Agility
User Stories	Product Owner	S	CAM	Agility
Wiki*	Developers	-	PROD	Internal communication
Analysis report	Developers	E,A	PROD	Internal communication
C4 diagrams*	Lead Developer	A	PROD	Internal communication
Validation Plan*	Tester	V	PROD	Quality assurance
Test cases*	Tester	V	PROD	Quality assurance
Regression Test Report	Tester	V	PROD	Quality assurance
Code Comments	Developers	-	PROD	Internal communication
DoD*	Scrum Master	-	PROC	Agility and Governance
DoR*	Scrum Master	-	PROC	Agility and Governance

* means the artefact is only used by P2T1. Code comments are referred to as 'notes in scripts' by P2T2. RE activity: E = Elicitation, A = Analysis, S = Specification, V = Validation, RM = Requirements Management, - = not applicable.
Type of artefact: PROD = Product artefact, PROC = Process artefacts, CAM = Core agile method artefact, CAM = Non-agile method artefact.

Table 4.4: An inventory of (but not limited to) documentation artefacts used by each team in P1.

Name of artefact (in English)	**Role responsible**	**RE activity**	**Type of artefact**	**Rationale of use**
Initial Contracts and Client Documentation	Client	E	-	Internal communication
Project Plan	Team Manager	-	-	Internal communication
Epics	Analyst	S	CAM	Agility
Analysis Report	Analyst	E, A	PROC	Internal communication
Scope Proposal*	Analyst	E	NAM	Internal communication
Functional Scenarios	Analyst	S	NAM	Internal communication
User Stories	Analyst	S	CAM	Agility
Wireframes	Analyst	A	PROD	Agility
External Interface Design	Lead Developer	A,V	PROD	External follow-up
Detailed Design	Lead Developer	A	PROD	Internal communication
Recorded Meeting	Analyst	E	PROC	Internal communication
Meeting Minutes	Analyst	E	PROC	External follow-up
Clickable Prototypes	Analyst	A, S, V	NAM	Agility
Product Backlog	Analyst, Team Manager	S	CAM	Agility
Functional Document	Analyst	S, V	PROD	External follow-up
Sprint Planning	Team Manager	-	PROC	External follow-up
Sprint Review report	Team Manager	V	PROC	Internal communication
Sprint Backlog	Analyst, Team Manager	S	CAM	Agility
User Interface Design	Analyst	S	PROD	Internal communication
Test Plan	Tester	V	PROD	Quality assurance
Test Report	Tester	V	PROD	Quality assurance
Wiki	Developers	-	PROC	Internal communication
Cases	Analyst	-	NAM	Internal communication
Programming Guidelines	Developers	-	PROC	Governance
Status Report	Team Manager, Analyst	-	PROC	External follow-up

Table 4.4: (continued from previous page)

Name of artefact (in English)	Role responsible	RE activity	Type of artefact	Rationale of use
Issues	Analyst	E	NAM	External follow-up
Change Management	Team Manager	-	PROC	External follow-up
Change Requests	Analyst	S,RM	NAM	External follow-up
Change (linked to user story)	Analyst	S,RM	NAM	Internal communication
Defects	Analyst	S,RM	NAM	Internal communication
SLA	Team Manager	-	PROC	External follow-up
Maintenance Guide	Developers	-	PROD	External follow-up
Release-related emails	Developers	-	PROD	External follow-up
DoD	Team Manager	-	PROC	Agility and Governance
DoR	Team Manager	-	PROC	Agility and Governance

* means the artefact is only used by P1T2. RE activity: E = Elicitation, A = Analysis, S = Specification, V = Validation, RM = Requirements Management, - = not applicable.
Type of artefact: PROD = Product artefact, PROC = Process artefacts, CAM = Core agile method artefact, CAM = Non-agile method artefact.

4.5 Conclusion

We explored two multi-team ASD projects using a multiple-case study approach with interviews and document analysis. The explored projects –P1: Education and –P2: Civil Engineering adopt scaled ASD differently. P1 uses an internally created method within the company, whereas P2 adopts SAFe. Furthermore, the processes of each team are divided into two phases, the requirements elicitation phase and the DevOps phase. The first phase ensures the elicitation and planning of at least the higher level requirements. The latter phase entails activities of the DevOps such as implementation, design, testing, and deployment. This phase is governed by Scrum for both projects. One of the common tools used by all the teams is Azure DevOps to keep track of the requirements..

RE activities are evident in the explored cases. For elicitation, techniques such as workshops and documentation analysis are used by the teams in P1, whereas P2 uses the planning recommendations by SAFe to guide the elicitation process. This entails the definition of a Roadmap and informal discussions with the stakeholders. The Roadmap entails epics, which are refined into features and user stories by the Product Owner. In addition to the models used by P1, all the teams in both projects make use of the refinement sessions to discuss any questions about the user stories. Also, the teams in P2 define an analysis task or research user story to conduct more analysis on complex user stories. In all the teams, requirements are specified using user stories which are stored in Azure DevOps. Various attachments in the form of links or documentation artefacts are added to a user story where necessary. In P1, user stories are reviewed and approved by the customer to validate the details of the requirement, whereas P2 takes the approach of discussing the user stories at the Refinement and Sprint Planning with the Product Owner to clarify any uncertainties. The process of handling requirements change in P1 is similar to that of P2, even though different artefacts are used. Usually, the existing user story is not changed, instead a new user story is added and the sprint plan is adjusted where necessary.

Chapter 5

Problem Investigation

From the explored cases in Chapter 4, we identified the various RE activities along with the documentation artefacts used. In this chapter, we discuss the results from the Problem Investigation phase of the Design Science Cycle within this research. Interviews and documentation analysis were conducted to identify the challenges facing the identified documentation as well as the best practices. We analysed the data of the findings and present a cross-case study analysis on the quality of documentation artefacts and practices. As discussed in Section 3.1, our research on the quality of documentation is centred around the taxonomy of documentation issues by Aghajani et al. (2019), as: *Information Content (What)*, *Information Content (How)*, and *Process Related* and the themes presented in Table 3.4.

5.1 Information Content (What)

For the assessment of the issues facing the content presented in documentation within the multi-team agile teams, we discuss the correctness, completeness, and up-to-dateness aspects of the quality of documentation.

5.1.1 Correctness

Preciseness of documentation

We asked the participants how the teams ensure the preciseness of documentation within the teams. A common answer was that it is *ensured during the Refinement*. In P1T1, the analyst asks the team members to review the user stories before the Refinement session, whereby questions are discussed among the team during the Refinement to ensure that the description of the user story is precise. 'Mainly by our Refinement sessions, [..] I would ask [the] team to read the description and to make sure they [..] are prepared. They might have questions, something might not be clear, but also before Refinement, we have a lot of discussions and checking in with each other' (Analyst, Interview 3). Also, *informal discussions before the Refinement are evident in P2T2 as well*, whereby the Product Owner discusses the user stories with the Domain Expert.

'Before the Refinement, [I discuss] it with the domain expert in the group. Then we have the Refinement and after that the Sprint Planning, everybody is asking questions. And then [..] we are making the [user] story [..] more detailed' (Product Owner, Interview 8). *Feedback from developers is also used to ensure the preciseness of user stories in P1T2*, 'I am also discussing with developers [..] to see if the technical side of what has been written down about the functionality is [feasible to implement]' (Analyst, Interview 4). Similarly for P2T1, the Refinement session is also used to clarify the details of user stories.

Another approach used by the teams in P1 is *the use of workshops with the clients to clarify requirements*. The analyst elicits user stories based on documentation analysis from the existing client documentation. Then, 'during workshops, we discuss everything of what I wrote down or that I copied [from existing documentation], to see if it is still relevant, if it is important, [and] what needs to be changed' (Analyst, Interview 4). Afterwards, *the team ensures that the details of the user story are precise by using the approval process*, whereby the Product Owner checks and performs 'the actual approval of the user story, if they [the client] say it is correct then we do not change anything unless we get approval by the Product Owner again' (Analyst, Interview 3).

Since evidence of other requirements documentation was found in the case studies, we asked how the preciseness of requirements documentation is ensured. Again this is *done during the Refinements* in the case of P2T1. For P2T2, the *requirements are made precise using discussions among team members*. Also, P2T1 *adopts the INVEST criteria when defining requirements*. Similarly, the *Refinement session* is used by the other teams, as well as *the approval process* in the case of P1 teams. On the contrary to P2, where user stories are not updated after implementation, P1 *updates requirements documentation after a user story has been implemented*. 'Once build is done and [UX designers] do a design and general functional check [on the] happy flow, [..] not nearly as detailed as testers do it, and then we update documentation. So once the new story is approved, we write it on functional documentation and add screenshots and make sure that, so if there is a change we really have to look back into the user story description' (Analyst, Interview 3). This also serves as a means to establish traceability of the requirements. However, this documentation artefact is not always updated after the implementation task of a user story have been completed. A reason why the functional document is not reviewed is due to the agile mindset, that the user story has a higher priority and that is leading. 'As long as we have the user story, that is the priority for everyone because then you can build something, what is written down in the functional documentation, it feels like an afterthought' (Analyst, Interview 4).

Furthermore, in the case of P1, the process of writing design documentation (software architecture and detailed design) depends on the user story, *documentation may be updated after the implementation task(s) of the user story have been completed*. 'Sometimes it is quite obvious what needs to happen and after we have developed the story, we update the documentation. Because the detailed design document is consisting of a use case diagram for a very basic schematic drawing of the functionality, it describes the functionality. So, it is not something that needs to be designed beforehand. [..], depending on the user story, if it is a larger one, [or] more complex one, then an architect or a lead

developer would make at least [..] a sequence diagram with it, so a developer has more context, [and] there is more information on how to build it' (Architect, Interview 2). Apart from that, we see a similarity in the review process, whereby *peer review is used by the teams in P1 and P2 to ensure the preciseness of design documentation.* An example of how this peer review process takes place for P2T1 is discussed. 'It is some kind of peer review process, so if you finish an analysis document, for example, you also have it read by the other developers, and the other developers can give comments on it. So, there they can [..] give some feedback, they can say what is well written and what has to be improved' (Lead Developer, Interview 7).

Adherence of documentation to the given templates

Adherence to the given templates of the various kinds of documentation within the project is also an essential aspect to ensure the correctness of documentation. The adoption of templates varies in both projects. For P1, there is a standardised template for documentation prescribed by C1M, whereas P2 does not have a fixed template for requirements documentation and user stories.

In P1, *user stories are mainly written by the analysts, which helps ensure that the given template is used.* 'I am the only one that writes down the user stories, so knowing that I already wrote down the templates, I just follow that template all the time. If there is someone else, I had sometimes some help from others, [and] then we review. Once they are done, it goes through me I review it and if there is something missing then I send it back to them' (Analyst, Interview 4). Also, *a copy is usually made from an old user story to the new one.* 'I usually start with pasting every header we need, [..] a design link header, [..] the functional description, [..] a technical description part, maybe some issues or questions that we have and other notes [or] anything. We usually define this list of what we want to add to user stories, and then I copy [and] paste this from the first line' (Analyst, Interview 3). Additionally, *as part of the training of new employees within C1, they learn how to use the templates and best practices available on the intranet of the C1M.* 'During the [training] they tell you about that. So, they say you have a document, and you have to fill in these things, so they prepare you to work like that' (Analyst, Interview 4).

In spite of the existence and use of templates for user stories within P1, there are challenges. *The template tends to get outdated due to constant improvements.* 'It is not up to date anymore, [..] the reason why is that we constantly try to improve it during Retrospectives, so I always ask the developers, what do you want to see or how can I make it better for you to understand? So, since we have been constantly changing it, the template is not accurate anymore' (Analyst, Interview 4). Also, *the Definition of Ready (DoR) defined for the project is not often used.* The reason for this is that most team members are not aware of the DoR. Similarly, the teams in P2 *have the Definition of Done (DoD) and DoR, but they are hardly used due to time constraints.* 'We have a [..] definition of done [..], but most of the time, we do not use it because of time' (Product Owner, Interview 8).

Similar to the responsibility of analysts in P1, the *Product Owners in P2 are responsible for writing up the user stories.* 'The product owner is responsible

for setting up everything according to this template' (Lead Developer, Interview 7). The problem is that 'there is a bit of a template, but [..] we do not always follow it. Actually, for most [developers], *we do not follow it, [..] because of time constraints*. And because of many of the other jobs that [have a higher priority], it is not always done like this' (Lead Developer, Interview 7).

For design documentation, the situation of both projects is similar to that of the requirements documentation. The structure of design documentation depends on the particular topic. The situation for P2T1 is explained as such. '*The design documents, so like [the analysis document] for example, [..] does not have a fixed template*. So, the only one that is really fixed here is [..] the C4 diagram, and for that, it is basically interpreted by the tool [..]. The analysis report or the wiki for example, there is not really a standard format. This can really depend on the topic' (Lead Developer, Interview 7). In comparison with P2 where there are no fixed templates used, the internally created method of C1 provides templates and best practices for various types of documentation used in P1. *The quality of templates and availability of best practices within C1M differs per type of documentation.* For design documentation such as the detailed design document, the template is pretty empty and there is a lack of best practices to learn from within the company. On the other hand, the template for the functional documentation 'could be very detailed, they have templates with headers and explanations [of] what they expect for each header. But in the end, the template is just templates and we use it for inspiration. I used to look into best practices to see what others did for this particular document and as best practices within [C1 ..]' (Analyst, Interview 3).

Examples of the most problematic documentation to keep correct

Before we elaborate on our findings with regard to the aforementioned topic, we add that these examples may be subjective as most of the given answers were based on the role of the interviewee within the project and what documentation he or she is responsible for. We elaborate upon the examples of most problematic documentation to keep correct in P1 and P2.

- *Documentation in general (P1).* 'Documentation is something nobody likes, and it is never a priority, which is understandable because, in the end, we need to deliver a product. [..], but it is [still] very important' (Analyst, Interview 3).
- *Detailed design document (P1).* 'I would say the detailed design because it is quite technical, time-consuming, and it gets outdated sometimes' (Architect, Interview 2). The main reason why it gets outdated is due to the fact that diagrams have to be updated manually. First, they are drawn using Draw IO, then exported as XML and PDF. Afterwards, they are copied to the document.
- *Functional documentation (P1).* 'Since I only work with functional documentation, I would say that is the most painful one [..]. Especially, because [..] after I'm done with [analysing] the user story, the developers will work on it and [..] eventually they say, we try to create it as you mentioned but we cannot, so we need to adjust it or the design needs to be adjusted a little bit, or there are change requests that cause the user story to be different or defects that changes the requirements, those kinds of things that I am pretty sure I forgot [to update] a lot of them' (Analyst, Interview 4).
- *User story descriptions (P2T1).* 'One of the most difficult things might be

the user story descriptions, and that is because even one description seems to be complete at first and after the implementation of a user story, it might actually turn out that this was not the correct implementation. Even though the implementation matches the user story exactly, it could be that after the [implementation], you notice [..] that it gives weird results for example. And that might mean that you have to rewrite the user story, or perhaps approach [it] in a completely different way. This often leads to some new user stories, and that might mean that we have multiple versions of the same user story' (Lead Developer, Interview 7).

- *Code comments (P2T2).* 'We do not have GitHub or versioning, [..] so what [..] we do [is], we are adding a date, the description, and the author. But if you are making a change and you forget to change the date, [..] it is not automatically, so it is all by hand. And people make mistakes. So, you can forget it. And then so that is the most problematic' (Product Owner, Interview 8).

5.1.2 Completeness

The extent to which requirements are documented

We asked the participants whether all the requirements are documented. *All the known requirements are documented for P1T1*, meaning there are no unwritten sources for the known requirements. 'We have a clear list of requirements, we usually refer to them in the documentation or the other way around, [..] we say this requirement is covered by user story x' (Analyst, Interview 3). Moreover, *in the case of P1T2, requirements are found in three locations.* There are user stories, requirements backlog, and business rules. Each of these sources of requirements serves a different purpose. In addition to the sources of requirements for P1T2, *user stories in Azure DevOps have certain attached documents in the form of links* which are outlined as follows:

- *Description of UI for the user story.*
- *Authorisation Matrix.*
- *Client documentation.*

Similar to the other teams, *the backlog is the main source of truth for the user stories in P2T1.* However, since the user stories are disjoint, *the wiki is used to collect central information about the requirements.* 'Requirements [can] also be found in the analysis report [or] from the wikis, especially because these user stories can be a bit disjointed. We also have, for example, this wiki to collect more of the information in one place instead of in disjointed user stories' (Lead Developer, Interview 7). In comparison, P2T2 takes a different approach to documenting requirements. 'For each user story, *[we (P2T2)] first have like an explore or analysis task, mostly it is a sort of POC.* So you make something and then you discuss if this is what we want to have, then we rebuild it and make it in [the] format which we have discussed with each other' (Product Owner, Interview 8).

The extent to which design is documented

We asked the participants if all the designs and tests are documented and whether there are some unwritten sources. *In P1T1, the architectural docu-*

mentation is well-documented for the most part. 'For the most part, we have documented everything because, we have an international team, it is important to document things in the first place' (Architect, Interview 2). *For the user interface design documentation, these are updated based on feedback from the workshops.* There is a log of the feedback but not on the changes to the prototype design, as the design might evolve a few iterations before it is accepted for the user story. 'I usually prepare wireframes for workshops and then they get feedback. Then, we document feedback, and then, we revise the design and then they approve it in the end by the user story, so we do not really have version control or anything on prototypes' (Analyst, Interview 3). However, in general, *the completeness of documentation appears to be solely reliant on the assigned person or role, but not the responsibility of the team as a whole.*

Furthermore, there was evidence of undocumented design in P2T1. *The team faces the challenge of finding the right balance between documenting and making the code self-documenting.* 'The main reason is that the documentation has advantages and disadvantages. So, the advantages are quite obvious I think, but also a disadvantage is that every time something changes and the person who changed it, might forget to update the documentation, in which case you get some inconsistency between the documentation and the actual code. And that is why when something is obvious from the code structure itself, we attempt to not document it, but we let the code be self-documented. But of course, a pitfall here could be that someone thinks that their code is self-documented, but then when reading it back later, it might actually be more difficult than people thought originally. So there is a bit of a fine balance here between doing documentation and keeping things self-documenting' (Lead Developer, Interview 7).

Additionally, we notice that *the design documentation in P2T2 lacks a clear file structuring making it difficult to keep all the various sources complete and up-to-date.* 'We have several folders on Teams, and maybe that is not a good spot but. And we have the Kennisbank, the knowledge side, which will which we want to expand upon' (Product Owner, Interview 8). In addition, *the 'Kennisbank' is used to document information about the solution of P2T2.* It is a knowledge platform being created to facilitate knowledge sharing about the solutions of the team with other stakeholders both internally within C2 and externally.

Accuracy and completeness of references used in user stories

We asked the participants how accurate and complete are the references (to documentation artefacts) used in user stories. *For P2T1, 'these are usually quite accurate*, so for example, they are often used for bugs. So if you want to reproduce the bug, then we often attach a save file in which that bug occurs, so you can easily reproduce [the bug] by simply loading that save file' (Lead Developer, Interview 7). Additionally, *if there is any missing documentation, these are discussed during the Refinement session.* 'Again, we ensure that those are complete using the Backlog Refinements. So, if there seems to be not enough information to reproduce something, we just ask for one of those lacking documents' (Lead Developer, Interview 7). On the contrary, *the references used within P2T2 may not be accurate as different static versions of the same documentation are used in the user stories.* 'In Azure, we can add a link to a parent or another user

story, and if I have a document, I mostly add it as an attachment. [But] that is not the best way to do it, [but it makes it] very easy for the user to have it all together as one package. [..] I think it is better to do it another way [..] it is really something that overview would help' (Product Owner, Interview 8).

In comparison with P2, *C1M provides an advantage to P1 by prescribing layout and landscape of documentation.* References are used within user stories as follows. In P1T1, the team 'usually links [the user story] to related tasks or related user stories, and not so much to documentation [..] it depends [..], but the documentation is in [SharePoint]. [We] follow [the documentation structure of C1M] so not a lot of new documents are added or get renamed or something' (Architect, Interview 2). For P1T2, the same benefits apply to the structuring of documentation. However, *the challenge occurs when the specification of the requirement or the authorisation matrix is constantly changing.* 'So the text [of the functional document] and the authorisation is always changing [..]. If there are textual changes while the user story has been completed, then it is a change request. So, you do not need to check the user story again' (Architect, Interview 4). Rather, in the case of change requests, new user stories are added.

5.1.3 Up-to-dateness

The consistency of the documentation with the working software

For this aspect of up-to-dateness, we asked the participants whether they perceive the documentation as consistent with the working software or product increment. There were a variety of answers, we begin with the strengths followed by the challenges as a potential for improvement.

In general, *the documentation of P1T1 is improved because the team had some time at the end of the release to catch up on documentation.* 'So, at the moment, it is because we had some more time and we actually working on it' (Architect, Interview 2). P1T1 attempts to ensure that the documentation is consistent with the working software. *There is documentation task(s) as part of a user story.* 'We have this task, documentation for every user story. So, the goal or the aim for it is to update [the] documentation with any user story that you finish' (Analyst, Interview 3). Additionally, *the four-eyes principle is used to ensure that the documentation task is reviewed by another individual.* 'There is a separate task to update the documentation and when necessary we do this. And the way we ensure the quality is that it is a task on the board, so it gets assigned to a person and [..] like any other tasks, the development task gets moved into a review column and then it gets reassigned to someone else. So, there is always a four-eyes principle that someone else reviews the documentation updates' (Architect, Interview 2). However, *the review mainly works for documentation whereby the responsible has a similar role as multiple people in the team.* For example, if a developer is responsible for a documentation task, it is likely that the task will be reviewed by another developer. But if an analyst is responsible for a documentation task, it occurs that he or she may have to review it on their own, because there is only one analyst on each team.

Despite the use of documentation tasks by the teams in P1, there are still some challenges. Firstly, *the decision to update documentation is at the discretion*

of the person implementing the user story. 'It is really easy to say that [..] for user story, we do not really need documentation, [but] usually you should [document]' (Architect, Interview 2). There is no process in place to discuss to what extent the team should document or what the acceptance criteria of that documentation task is. So even though there is a documentation task, the responsible assignee may move it to 'completed' without actually updating the documentation. Also, *team members are not always motivated to work on documentation*, so the documentation task tends to be skipped sometimes. 'We always have to check [the documentation task] at least. And like I said, it does not always happen, people just do not feel like doing it' (Analyst, Interview 3).

Moreover, *the process of updating documentation is still a learning process for the teams within P1*, especially since change requests hardly include documentation tasks. 'That was a learning process for us to not forget [that,] once we have a change request or defect, we should document that as well. And we did not do that for a very long time, so [..] the new things that we are doing [are] documented, so that is up to date. But all the things are not up to date, so the document will never be perfect' (Analyst, Interview 4). Also, since tasks may be completed or moved along the columns of the sprint board, *there is no process in place to review a user story and assess if it is really done.* 'How it happens is because we do not [review] a user story and [we] do not follow strictly the definition of done. Like there is usually [no] other formal moment where we look at our user story and say OK, is it really done' (Architect, Interview 2).

In comparison with the teams in P1, *P2 do not define a documentation task as part of the user story.* The *scope of documentation per user story is not clear* and *there is no formal check in place to ensure documentation is updated.* As mentioned earlier, P2T1 deals with the challenge of finding the right balance between documenting and keeping the code self-documenting. 'That is why when something is obvious from the code structure itself, we attempt to not document it, but we let the code be self-documented. But of course, a pitfall in here could be that someone thinks that their code is self-documented, but then when reading it back later, it might actually be more difficult than people thought originally' (Lead Developer, Interview 7). P2T2 have only been working on the project for nine months, so at the time of interviews, *the documentation was perceived as 'sort of' consistent with the working software.* However, *there are concerns for the future as the project expands.* 'But it is easy to say after nine months or something and [P2T1] are working on it for 4 years, and in four years many things will change, and then you will get this situation. I think that is more difficult, but that is also because the application grows. Sometimes you are making, a feature and you make it documentation, and then after a while, you add something to the application, and the additional parts have an effect [..] on everything and the documentation stays the [same], so sometimes it [gets] outdated' (Product Owner, Interview 8).

Examples of functionality in the code that is not yet documented

As evidence to support the previous question about consistency, we asked the participants to recall any examples of functionality in the code that is not yet documented. We discuss the examples.

- *The frontend, due to a large refactoring (P1T1).* 'Once you get into this habit of not updating the documentation, it is really easy to postpone it. And we spent a lot of time actually refactoring existing code, which is part of the reason why we ran out of time. We built a frontend, but we build it in the wrong way, so we spent a lot of time refactoring the frontend. And [..] if we did not have to spend all the time refactoring, we would have had a lot more time for quality, and so for documentation' (Architect, Interview 2).
- *The functionality to create a new declaration is not documented correctly (P1T2).* 'There was a functionality [..] instead of creating a new [declaration], you copy all the information into a new page. I do not think that is documented correctly, that it is now copying instead of creating a new one. [..] So, the software [..] works and the guidelines that they have for the users are all updated, but the documentation that we have is not' (Analyst, Interview 4).
- *The frontend and the functionality of the design margins (P2T1).* 'The frontend is not very well documented in general. [..] It is something called the design margins, and it is generally not very well documented. So, those could be two examples like the frontend and design margins [..] there is hardly any documentation about the frontend' (Lead Developer, Interview 7).
- *Nothing in particular, but more of a general perception (P2T2).* 'I cannot [recall] it, but I think [..] we need to do it better, but that will take more time. So, it is maybe it is just a feeling [..] we want to have progress, and if you want to go too fast, you do other things less, so the documentation is maybe too little' (Product Owner, Interview 8).

The extent documentation is sufficient to extend or maintain the application

Here, we asked the participants whether they perceive the documentation to be sufficient to extend or maintain the application. We present the findings beginning with the strengths and the potential areas we identified for improvement.

The general perception of the teams within P1 is that the documentation is sufficient to extend or maintain the application. 'At the moment, yes [it is sufficient], I would say there is definitely room for improvement [..]. The detailed design document is fairly extensive, so we have use cases. We have those sequence diagrams for almost all of the functionality now. There is an explanation of how the backend code and the frontend also are structured. So basically, the files on disk, and how the files are stored in a file structure. And the reasoning behind it, especially the frontend framework that we developed, is documented. So,[..] if a separate company needs to take over this application, I think they are pretty well served' (Architect, Interview 2). Also, *from a functional perspective, the documentation is perceived as sufficient.* The new requirements and change requests to the original scope of the project are well documented. 'I would [say we] do have to keep track of [the] functional documentation because we do get change requests' (Analyst, Interview 3).

In the case of P2T2, *a strength is the development of a knowledge bank*, to provide other stakeholders with knowledge about the application by providing videos and also organising workshops. 'That is the part where you want to have small movies, so the workshop is to first know the program, and then to know the tooling. And the next step is to use our tooling [..] and that [..]

takes some effort to get there' (Product Owner, Interview 8). However, since the knowledge bank is still in progress and the documentation is not sufficient, *P2T2 faces the challenge of not having sufficient documentation to extend or maintain the application.* 'I think that the documentation is too small. We can document more, but that will also take more time, so [..] there is a balance between making things and documenting things. And our focus is more on making things, and that is a bit of a risk' (Product Owner, Interview 8).

Moreover, *P2T1 does not have sufficient documentation to extend or maintain the documentation and that is mainly due to the challenge of documenting and making the code self-documenting.* The risk of that is the knowledge of some parts of the application are strongly reliant on specific experts within the team. 'At the current moment, I definitely do not think it is sufficient. So, I talked before about this balance between having documentation. But also, to avoid inconsistencies. You also do not want to have too much documentation, but I think on the sliding scale we are too much on the side of too little. And [..] you can notice this. For example, [..] the current development of the application is quite dependent on specific persons' (Lead Developer, Interview 7).

The extent to which the translation in the documentation is outdated

We asked the participants whether translation is used for documentation and if it is outdated. *The teams in P2 do not have translation issues as P2T1 works entirely in English and P2T2 works in Dutch.* Each team agreed on a common language for the team. *For the teams in P1, they work in both English and Dutch.* English is internally used as the teams are made up of different nationalities, whereas Dutch is used to communicate with the customer. This provides a challenge as to what documentation should be in Dutch and which ones can be in English. Also, it means for the Dutch documentation, not all team members are able to review and update the documentation.

There are no translations for the requirements documentation for P1T2 that are outdated, but both versions have to be updated simultaneously. 'Because [P2T1] just started translating it this year. So, only the new user stories that we have are in Dutch and English. And once I edit, I adjust the Dutch immediately added [and] adjust the English version as well' (Analyst, Interview 4). Furthermore, *it is not clear whether to keep the technical documentation in English or Dutch.* 'We mostly translated the documents to Dutch. Actually, the preference is to keep them in English, but [..] it is still a bit unsure if the technical documentation can be in English or not [..]. I know there is a software architecture document. [..] currently, it is part in English, part in Dutch, and still, the decision needs to be made [whether to have it in Dutch or English]' (Architect, Interview 2).

The challenges faced when ensuring documentation is kept up-to-date

We asked the participants about the challenges faced when ensuring documentation is kept up-to-date. There is a broad spectrum of reasons, we discuss them as follows.

- *The increase of documentation debt due to the deadlines of the sprints and the focus on delivering a product increment at the end of each iteration (an agile*

principle). 'So, in some cases we cannot finish the user story in a single sprint. And then, the documentation task or some other additional tasks get moved to the next sprint [..]. Sometimes, it falls through the cracks and then when we near a release period like right now, we have a week or two where we need to write a lot of documentation. So, in the end, it does get done because part of the non-functional requirements is that we have up-to-date documentation' (Architect, Interview 2). Also, the time constraint is evident in P2 as well. 'Time constraints can often be a thing, like if there is a product deadline, then, something will have to be scrapped from the planning. And very often stuff like the non-functional requirements such as documentation gets scrapped the earliest' (Lead Developer, Interview 7).

- *Updating the UML diagrams in the detailed design documentation is inconvenient and time-consuming.* 'Our challenge is the fact that we use Draw IO, and it requires discipline [for the] team [to] update the Draw IO file because you have to download it to your computer. So, when you make changes, then you upload it again to [SharePoint], and then you generate a PNG file and update it in the Word document. So, in that sense that is quite a challenge that it is updated in that way' (Architect, Interview 2).

- *The motivation of the team members to update documentation is lacking.* That is mainly due to the goal of the sprint in line with the agile mindset to focus more on the working software rather than writing comprehensive documentation. Hence, 'people just do not feel like doing it [writing documentation]' (Analyst, Interview 3).

- *Documentation gets forgotten and managers have to enforce the process of updating documentation.* 'Right now, we see those tickets [documentation tasks] stay in the to-do [column] and then the managers [enforce that the team] have to do it, so we will not continue with the new user story before you finish documentation then we would just assign it to people' (Analyst, Interview 3).

- *For the functional documentation, review with the four-eyes principle is lacking.* 'Review [..], once it is written down, we had reviews, and we do not have that any more' (Analyst, Interview 4). In this case, the problem is due to the downscaling of the team. 'When we were with 2-3 analysts, [..] once you are done with the documentation, that [was reviewed] by someone else. But now because I work alone, it is difficult' (Analyst, Interview 4).

- *It is not clear what documentation needs to be updated.* 'Knowing when documentation has to be updated. Because some changes in the application require updates in the documentation and some of them do not and beforehand you might not always know if something requires an update of the documentation or not' (Lead Developer, Interview 7).

- *Determining what and how much to document is at the discretion of each individual.* As mentioned earlier in Section 5.1.3, the decision to update documentation within P2 is at the discretion of the individual. 'Within the team there are some varied opinions on where you should be within that spectrum, that makes sense because it is a spectrum. So, we do not really have any formal requirements on when to document and when not to document. So, in this case it is mostly at the discretion of each individual' (Lead Developer, Interview 7).

- *There is no check in place to remind developers to update the documentation in P2.* 'I think there is no such reminder in place at the moment, so at most, it could be that during the Refinement we might see that some update to the documentation is needed and that would just mention it within the user story

as well.' (Lead Developer, Interview 7).

- *The limited capacity of P2T2.* 'A challenge is the capacity, so now we have four people [..], and they work part-time so and at the end, we have two FTEs (full-time employees). But [..] we want to have more progress and we have much work to do, so capacity is a challenge' (Product Owner, Interview 8).
- *The process of acquiring projects for P2T2.* Functionality within the project is in line with the project obtained by C2. 'And in projects, what we see in The Netherlands is that [..] sometimes we have a project and it doesn't start because they are not allowed because the sometimes [the contractor] [does not comply with legal regulations]. And we have several projects and so the contractor [obtain their] goals first [..], but if the project starts, they do not need us anymore because [..] several projects are stopped' (Product Owner, Interview 8).

The difficulty in tracing the versioning and updates of the documentation

We asked the participants how difficult is it to keep track of the versioning and updates of the documentation. In general, we found a number of good approaches in how the versioning and updates of the documentation are kept up-to-date to ensure traceability. However, we also found some challenges.

The internally created method of C1, *C1M, provides templates for documentation which also includes guidelines toward keeping track of versions of the document.* In the template, 'there is a legend for it, but people do not fill it in always. And you can cross-reference documents whereby you can [refer] to a specific version of another document' (Architect, Interview 2). However, *even though the template guides writers to fill in the versioning and references, it tends to be forgotten.* 'That is also not being updated [..], because it is not formalised in any process' (Architect, Interview 2).

Moreover, *adding versioning to documentation is taught during the training for new employees within C1*, so they should be equipped to incorporate it in the process of writing documentation. 'During the [training] they tell you about that. So, they say you have a document, and you have to fill in these things, so they prepare you to work like that' (Analyst, Interview 4). However, even with this knowledge, not everyone remembers to update the versioning and reference tables in the document.

Nonetheless, *user stories in Azure DevOps for both P1 and P2, automatically provides information about the history and changes made to the user story.* For P2T1, *the versioning of changes made to the wiki is kept track of automatically.* 'This is mostly done automatically. For example, the wiki has an automated system for tracking changes. It is very much like Wikipedia, which always has [a] revision history, the same with Azure DevOps. And with the analysis report, for example, we used the word functionality for tracking revisions [..]. So for this we really do not have to do much as a team, that is really all done automatically, which is quite convenient' (Lead Developer, Interview 7).

P2T2 on the other hand has to *manually update the versioning by means of code comments in their scripts.* 'But if you are making a change and you forget to

change the date, [..] it is not automatically, so it is all by hand. And people make mistakes, so you can forget it' (Product Owner, Interview 8).

5.2 Information Content (How)

For the assessment of the quality of the information present in the documentation artefacts used within the multi-team agile teams. We discuss the maintainability, readability, usability, and usefulness of the documentation.

5.2.1 Maintainability

The difficulty in adding changes to the requirements documentation.

As the first aspect, we consider for the maintainability of documentation within the team, we asked the participants whether it is easy to add changes to the requirements documentation and if it is clear to know what other documentation artefacts to update in this case. We discuss our results.

For both teams in P1, *it is fairly easy to add changes to the documentation, this is because C1M provides a landscape of documentation and the templates also include a reference table that indicates what other documentation is referred to in a particular documentation.* 'There is [..] the landscape of [C1M] documentation. Because they are all connected in a way and it is like we have connections in LinkedIn, like this web collection or connections, there is also a web of documentation.' (Analyst, Interview 3). Also, *the documentation prescribed from C1M and used by the team are defined with a clear purpose.* 'It is fairly easy to add changes. Again, we followed [C1M], the documents there have a clear purpose which really helps. Some have like a standardised format, so a standardised list of chapters' (Architect, Interview 2).

Furthermore, *most of the documentation used by the teams in P1 based on C1M are Word documents stored within the SharePoint environment of the team.* 'The documents that we deliver are pretty easy [to change] because they are in [SharePoint]. So, it is in Word and then you can just edit, everybody has access to it. The good part also about [SharePoint] is you can see who modified for the last time, who checked it out and you can even go back to previous versions. In Azure DevOps the user story changes are also pretty easy, everybody has access there' (Analyst, Interview 4)

In spite of the benefits C1M provides to the documentation of P1, *it is still not always clear what other documentation needs to be updated.* 'No idea on that (knowing what other documentation to adjust), that is a really good question, [..] I find myself pretty lost [about] what we all need to document with those kinds of things [..]. And it depends also on the change request, so, if it is a validation or [..] a business rule or requirement, I will reconsider checking then one of the places that we fill that in. But quite often, things are very specific so, you do not need to adjust on those things, but only on the user story. So that is my part, but I sometimes think also it should be updated somewhere else as well, but [..] if it is technical, then I rely [..] on my team to know' (Analyst, Interview 4).

In P2T2, *the scripts are tightly coupled to each other and also the comments are updated manually.* Since there is *no clear overview of the landscape of documentation*, a change affecting multiple documentation is difficult to address. 'If we change one thing, we must do it in every document so it is not that simple. And this we have to do it manually per document, so [..] it is challenging' (Product Owner, Interview 8).

On the other hand, *the issue facing P2T1 is not about having documentation to update in different sources or duplicates, but rather, the team has outdated documentation.* 'You could of course have like more of implicit duplication in the sense that you always have [..] the code itself and its documentation and those could not match with each other, because like the code describes the same thing as the document. In that case, it is more of an implicit duplication' (Lead Developer, Interview 7).

The difficulty in adding changes to the design documentation.

With reference to the design documentation within P1T1, *some of the documentation are pretty large, making it difficult to find the right place to update.* In this case, it will require knowledge of the person who wrote the document to figure out which part of the documentation to update. 'It might be difficult to find [the] right place in which you need to update it [the functional document]. Because I think the document we have now is over 80 pages. And I wrote it myself, so I know exactly what's in what chapter. But that is what we used for user story references. So, if it is a change, it is usually linked to a user story, and if it is not, I somehow know I remember what part of the functionality is from. So, you just go to that part of the document' (Analyst, Interview 3).

For P2T1, the level of difficulty in adding changes to the design documentation is the same as that of the requirements. Basically, *they try to eliminate duplicates as much as possible to make it clear and easy to update documentation.* 'The documents there also should not be [of] any duplication. So if there is, we just address it and move on' (Lead Developer, Interview 7).

The awareness of how a change in one documentation impacts the others.

For this topic, we asked the participants whether there are processes in place to know how a change in one documentation impacts the others. The results are discussed as follows.

In P1, *sometimes the user story also mentions what kind of documentation to update.* 'Basically, we have those workshops with the customer and we have Refinements and during the Refinement estimation session, we specified [..] what kind of documentation do we need to update' (Architect, Interview 2). However, this is not always the case. *Even though it tends to be asked during the Refinement, it is not formalised in the structure of the meeting so it tends to be forgotten sometimes.* 'Sometimes when you are picking up a user story, to know which documents you need to update, it is not always clear, that should be part of the template for [the] user story on Azure DevOps. But when we have a Refinement session, we ask the question, 'which documents need to be updated?'.

That is good [and] very helpful. So, we need to formalise the procedure for those kinds of meetings' (Architect, Interview 2).

Similarly, in P2, *it is not always clear how a change in one documentation impacts another.* 'This is definitely not always clear so that you have to know [..] which things are documented and which are not and also where they are documented. That is very much a team responsibility, so if someone thinks of documentation that is associated to something, you should mention it during Refinement session for example so that we can make a note that it has to be updated' (Lead Developer, Interview 7).

The challenges of incorporating change requests in the documentation.

We investigated the challenges of managing requirements volatility. We asked the participants about the challenges their team faces when working with changes in requirements. There was one best practice from P2T1, which we discuss as follows and then, we elaborate upon the challenges we identified as well.

The documentation in P2T1 is maintained with the principle that *documentation is made modular.* If the documentation gets large, the team splits it into smaller parts. 'We usually try to split up into smaller pieces. So not trying to do everything at once, but split it into smaller pieces, where each of those pieces is a bit simpler to do. [..] otherwise, you might just lose the overview because indeed there could be something that is described both in the user story and in the wiki. So one [..] from the requirements point of view, and one [from a] design point of view, but sometimes even though different points of view, they might still describe the same thing, just from different viewpoints, both have to be updated. But splitting everything into small pieces you restrict the number of places that have to be updated as much as possible' (Lead Developer, Interview 7).

The challenges facing the incorporation of changing requirements on documentation is highlighted as follows.

- *Limitations of the tools (P1).* 'The [SharePoint] does not always work, that is the downside of it, [..] so I could not save it to the cloud. I could only save it locally. So the connection is not stable enough, and sometimes if those things happen sometimes people save things locally, [..] and people forget to upload it soon. There is not always the latest version in [SharePoint], and if someone else changes it, you get changes or version conflicts and that is a problem' (Analyst, Interview 3).
- *It is not communicated clearly what needs to be updated in the documentation (P1).* As already discussed in this section, this is also a challenge that hinders the effectiveness when incorporating changes to requirements in the documentation. 'With user stories, we have this template. [..] We have this fixed list of tasks we add to user stories, that is different [from change requests]. For instance, now in this phase of the project, we are in this functional acceptance test phase for [the] customer and there are change requests, but I do not get notifications to change functional documentation, so it is the release process is very clear and after the release, once the change request process starts [..], then documentation is forgotten' (Analyst, Interview 3). Also, 'one of the things is also the content. I think that we have not discussed what is truly necessary to document, [and]

what are things that you do not need. Because [..], I have been struggling a little bit sometimes' (Analyst, Interview 4).

- *Even though there is a formal process in place to handle change requests, the documentation aspect is not enforced (P1).* 'There is [a formal process for change requests], but I do not think we [..] use it or, change requests are usually small bits of things that we need to change that, it is not necessarily a big chunk of functionality or a lot of hours. We try and think it is just ticking checking boxes' (Analyst, Interview 3). Due to the need to implement those small functionalities quickly, the documentation tends to be forgotten.
- *The code comments in the scripts are updated manually (P2T2).* When new user stories are added, the documentation has to be adjusted manually. As P2T2 makes use of code comments to track versioning and changes, these have to be maintained manually in all dependencies after a change has been implemented. 'We add new stories, and we make it as a note in the code by hand. [We document] the author and the date and what is changed, but it is all by hand' (Product Owner, Interview 8).

Addressing the question of what is enough documentation.

As we noticed from the interview, there was an occurring concern that it is difficult to know what is enough to document. Therefore, we asked the participants how they address the question of what is enough documentation as a team. We discuss the two recommendations.

Firstly, P1T2 addresses this by means of *feedback from the various expertise within the team.* At some moments within the project, the analyst asked for feedback from the team on the user stories. 'That was only a few times at the start of the project, but also halfway through. [..] the first time was just in general, the analysis team came together and they [discussed] how are you going to write the user story, what is the expertise, what do we need, what is essential, what kind of headlines do we want, and from there we [discussed] what do we need for the functional documentation. And then, because you have more people with different experiences and [skills within] in this project, [some said] I thought it was really nice to see links to the user story inside the functional documentation because then you can check and click and you know it is accessible, then you add that to the list. Others say, well, I like that it has numbers so you can discuss chapter 1.1.1.1.1, and then someone will know where it is. And that is how then we organised it. And there was for me it is also important [..] after [we had] done it. So that is why after a year, I discussed with the rest of the team, how are you feeling now with the document, are there still things missing also with the user stories, basically asked them for their feedback on the documentation' (Analyst, Interview 4).

Similarly, *P2T1 tackles this challenge mainly via informal communication among the team.* '[It is] mostly [done] via informal discussion. Within the team there are some varied opinions on where you should be within that spectrum, which makes sense because it is a spectrum' (Lead Developer, Interview 7).

5.2.2 Readability

Processes in place to review documentation and ensure that they are clear to read.

We asked participants whether there are processes in place to review documentation and ensure that they are clear to read. We found some strengths in how the teams approach this aspect, but also, depending on the type of documentation, there are some challenges. We first elaborate upon the strengths and then discuss the challenges as potential aspects for improvement.

- *Feedback on user stories is obtained during the workshops with the customer (P1).* Feedback on the clarity of the details of the user story is attained through the workshop and the approval process with the customer after the workshop. 'We have the workshops as a result of that, we define the user stories and the user stories get accepted. So, we do have a meeting for that, [..] it is per user story or we take a bunch of user stories and go through them and then you know the customer can give feedback' (Architect, Interview 2).
- *Peer reviews among team members (P2T1).* 'This is mostly done through peer review [..]. For example, the requirements are checked during the backlog Refinement. Where every developer can have to look at some readable for the design. Usually before publishing something [you have it] read by other developers [and they] give feedback [..]. In that way, we try to ensure that is a good quality' (Lead Developer, Interview 7).
- *Using the Scrum events (P2T2).* 'We do a lot of talking. So, we discuss it in the Refinement or [Daily Scrum]. And so [..] we are using our work process' (Product Owner, Interview 8).

With that, we discuss the problems hindering the review of documentation.

- *There is no review in place for the functional documentation (P1).* As mentioned earlier in Section 5.1.2, feedback is mainly given on documentation within P1, when there is more than one occurrence of the role of the responsible, for example, the role of a developer. Hence, documentation such as the functional document or analysis documents do not get reviewed by other team members. 'For us the analysis team, we do not have reviewers.' (Analyst, Interview 4). 'We usually have one person writing it and then assigning it to the other person who can review it. If you are the only analyst that is hard, so then you have to just trust yourself basically' (Analyst, Interview 3).
- *Documentation is rapidly reviewed and updated nearing the end of the release (P1).* 'At the moment we do not have like a recurrent task for this, so for [the product of the team] we are nearing the production years and because of that we are going through all the non-functional requirements, and as a result, we are updating the documentation, so that is one in one moment, at least when we are looking at documentation' (Architect, Interview 2).

The conciseness of the contents of the documentation.

To assess whether the contents of the documentation within P1 and P2 are concise, we asked the participants for their opinion. Due to the lack of feedback from both external and internal stakeholders, this is quite difficult to determine. We discuss the arguments.

Firstly, in the case of the teams in P1, *without adequate feedback, it is difficult to ensure conciseness of the content of the documentation.* For the analysis documentation of P1, they have not received any feedback from the target readers. 'I have not [had any feedback from the client] yet. I am not even sure if they have checked documentation' (Analyst, Interview 3). Also, for the technical documentation as well, they do not get any feedback from any reviewers outside the team such as the client. This is because the client does not have the necessary technical skills to review this documentation. 'Because I do not think [the customer] reviewed them. This customer, [..] does not have a lot of technical knowledge' (Architect, Interview 2).

On the other hand, for P2T1, *the level of conciseness depends on the complexity of the topic being written in the documentation.* 'It really depends not on the topic, so usually, the more complex the topic is, the more documentation it requires' (Lead Developer, Interview 7).

5.2.3 Usability

Issues faced when using the various tooling to write documentation.

As discussed in Section 4.2.3, there are various tooling used by the studied cases in writing documentation artefacts. We asked the participants what issues they face with the tooling with regards to writing documentation.

There are a few challenges facing the use of SharePoint for P1. Firstly, *the performance of the tool*, the '[SharePoint] is really slow' (Architect, Interview 2). Also, we see evidence of *connectivity issues* sometimes which in turn causes problems such as merge conflicts when the connection resumes. 'The [SharePoint] does not always work, that is the downside of it, [..] so I could not save it to the cloud [..]. So the connection is not stable enough, and sometimes if those things happen sometimes people save things locally, [..] and people forget to upload them soon. There is not always the latest version in [SharePoint], and if someone else changes it, you get changes or version conflicts and that is a problem' (Analyst, Interview 3).

For the teams in P1, *even though the documentation is in SharePoint or Azure DevOps, feedback from external stakeholders is gained via email, and then the document has to be updated.* 'For the change request, [..] last time what happened is that we sent a link to the document in an email, and they email back saying it is OK, and I think the customer is able to edit the document themselves and change the status to approved. But [..] usually, we do it for them' (Architect, Interview 2). With this approach, the feedback cannot be traced to a central location.

Furthermore, one of the challenges P2T1 faces occurs with the use of PlantUML. 'The C4 diagrams are a little more difficult because *PlantUML is a bit of a*

restricted language, so sometimes there could be things here that are not possible within that language, just because it is not very expressive' (Lead Developer, Interview 7).

The level of difficulty for external stakeholders to find information in the various documentation.

We asked the participants about the difficulty for external stakeholders to find information in the various documentation. There are two best practices we identified with regard to making information accessible to the stakeholders. Besides, a general remark was that there is no feedback from the external users yet with regard to this. We discuss these findings.

The documentation structure based on C1M is standardised across projects within the company. Therefore, 'once you are familiar with [C1M] and you know about the different kinds of documents, [..] it is easy to find them' (Architect, Interview 2). However, *P1 have not received any questions about documentation from external stakeholders as yet due to the skill set of the client.* 'I do not think [the customer] reviewed them. This customer, [..] they do not have a lot of technical knowledge' (Architect, Interview 2). 'I have not really heard from the customer a lot about the documentation yet, [..] about the deliverables, no, I have not really had that [many] questions, and they either did not read it or it is very clear' (Analyst, Interview 3). *In P2T1, we also see a similar remark.* 'I have rarely had those types of questions. I do not really know why, I would personally expect a lot more of those questions, but I have [..] rarely had someone [..] asked for documentation' (Lead Developer, Interview 7). Instead, if there are questions, external stakeholders tend to ask them in the form of emails or via Teams.

On the contrary, *P2T2 receives questions from stakeholders outside the team, specifically within their department.* Due to the limited capacity of the team and the inefficiency in giving the same explanation a number of times, *P2T2 decided to address this issue by creating the 'Kennisbank'*, this is a knowledge platform with videos. 'Yes, I think it is [difficult for other stakeholders to find information] and that is why we have made the Kennisbank, to share the information we want to share. Because we do not want to to share everything, [..] that is too much, we want to share the only the parts we [can] share [at the moment]' (Product Owner, Interview 8).

Processes in place to ensure shared documentation is maintained correctly by both teams.

Having a multi-team project, we were interested to know how the teams collaborate to maintain shared documentation. We discuss our findings.

As both projects consist of teams that are divided based on functionality, the shared documentation among teams is usually on a higher level. Within P2, *documentation is mainly shared on a higher level, such as features and program increments.* 'Between [P2T1] and [P2T2], it is mostly at the high-level overview, so not at the level of design or testing, but at some of the requirements level, not at the level of user stories, but really at the level of features and the program

increments. That is the only place where the connections between two are documented' (Lead Developer, Interview 7).

Similarly, *a high-level functional documentation artefact is shared by both teams in P1.* Each team has their own functional document, and they share a global functional document that outlines the common components and functionality within the team. *However, this is not updated by both teams simultaneously, rather, a responsible is assigned to maintain the document.* The analyst in P1T2 is responsible for updating the document. Hence, if there are changes that impact the shared functionality, they should be communicated properly. documentation 'It depends if it is a change in [P1]. [..] if it is a global change and they [the other team] need to be aware, then it is in the global document' (Analyst, Interview 4).

Despite the benefits of the alignment meetings across the teams in P1, *the meetings hardly incorporate, discuss or review documentation.* 'Earlier, we talked about this design analysis [meeting] with the designers or the interface consultants. We have this weekly meeting in which we discussed the component sharing part, but [..] we only use it for wireframing and prototyping. And I am not sure if they do it on a documentation level' (Analyst, Interview 3).

5.2.4 Usefulness

Processes in place to encourage the various stakeholders to review and give feedback on the documentation.

To understand how useful the documentation is to the various stakeholders, we asked the participants whether there are processes in place to encourage the various stakeholders to review and give feedback on the documentation. We discuss our findings.

In the case of P2T1, *that depends on the stakeholder.* 'The design documentation is mostly used among the developers. So, it is really documentation by developers for developers, so it is not really read very often by external stakeholders. And the requirements depend [..] on how high-level it is. So the low-level requirements [..] mostly stay within the team because it is often a bit too low-level. For example, managers do not look at [..] the user story exactly, but the more high-level requirements, so [..] the level of features or epics or the program increments [..]. So this could for example also be discussed together with other users to see, where they would like to see the program go and this is also interesting for managers to make a bit of a high-level planning' (Lead Developer, Interview 7).

In turn, *P2T2 and both teams in P1, do not receive feedback from external stakeholders, and the process of encouraging feedback is absent within the teams.* For the technical documentation, 'There are no processes in place, so we do not periodically ask [the client] for feedback, we could do that. It would really improve the usefulness' (Architect, Interview 2). Also for the analysis documentation, we see the teams do not receive any feedback. 'I do not hear [the client] discussing or reading or approving the documentation, so I feel like it gets forgotten, not only from our part but their part as well' (Analyst, Interview 4). Also, *for P2T2 they do not get feedback until they use the solution within an engineering*

project. 'We do not [..] have feedback yet from stakeholders. But sometimes we [use] a small part [of the solution] in the project and then we get feedback. [..] But we do not get much feedback' (Product Owner, Interview 8).

According to the Architect, *a reason why there is no review process in place in P1 for external stakeholders is that it is not part of the DoD.* 'If it was a part of the definition of done that the documentation needs to be updated, and the customer would do a critical review, then that would be the formal process. It is not part of the [DoD], we do not really follow the definition of done very strictly, and the customer does not critically look at the deliverables' (Architect, Interview 2).

Planning and incorporating feedback in the documentation process.

In the case of feedback on documentation, we asked the participants about how feedback is planned and incorporated into the documentation process. Our findings are discussed as follows.

Within P1, *there is no process in place describing how feedback or issues should be handled.* However, the teams incorporate a number of approaches to address feedback on documentation. Firstly, *the Daily Scrum may be used to discuss any feedback or issues with documentation.* 'If someone spots an issue in the documentation, you can of course always mention it during [the Daily Scrum] or otherwise' (Architect, Interview 2). Then, *the functional document is approved at the end of each release.* 'At the end of the project, [the customer] approves user stories on a functional basis. But they do not approve the document each week, they do it at the end of [..] the release' (Architect, Interview 2). After documentation artefacts such as the Functional Document or the Minutes of Meeting are sent to the client for review and approval, communication is mostly done by email, the feedback is incorporated into the documentation as soon as possible.

The aforementioned situation is similar to that of P2. *When there is feedback, the team tries 'to update it this quickly as possible.* When feedback [is given], if you agree with the feedback, you can just immediately update it. If you disagree then you just discuss it with each other to try to come to some kind of agreement. But, to prevent these processes from taking too long, we often try to keep the time between the feedback and the update as soon as possible' (Lead Developer, Interview 7). Also, *on a high level when there are comments on a feature, the product owners of both teams discuss it together.* In future, they intend to address this using an issue-tracking system. 'If you have any feature in [which] we have much feedback. Then you have to process this, [..] I have talked to [the Product Owner of P2T1] about it and we have discussed the ticket system, and in the future we [..] must process the feedback with tickets [..], because you have to prioritise them or you have to think about what is cheap and more gains [..]. So we are thinking about the process about the feedback [process]' (Product Owner, Interview 8).

5.3 Process Related

For the assessment of the challenges facing the processes of writing documentation within the multi-team agile teams, we investigated the challenges of internationalisation, contribution to documentation, automated documentation and other issues facing documentation. Lastly, we conclude with the strengths or best practices used in documentation practices within the teams.

Internationalisation

We asked the teams whether there are difficulties faced during the translation of documents. Teams in P2 do not need to translate documentation, P2T1 works entirely in English, as the team is international and P2T2 works in Dutch, as the team consists of only Dutch speakers. Teams in P1 on the other hand, face challenges with translating documentation artefacts and keeping these aspects up-to-date.

Firstly, to address the language barrier, both teams, P1T1 and P1T2, *conduct the activities and Scrum events such as the Daily Scrum, Sprint Planning, Retrospectives, etc. in English.* 'I do think that I do everything in English. The code, [..] chats are in English, meeting chats, [Daily Scrums], Retrospectives, Refinements [are] all [in] English, all team activities and meetings are English' (Analyst, Interview 3). *However, there is a language conflict between what the client expects and the communication of an international team.* 'It is really challenging because our customer requires Dutch documentation and also user stories. And actually [as the] language during workshops, they do not really like doing it in English, which I understand, [..] like the terms as well, the domain knowledge you do not always have a translation. So it is hard to do it in English, but it really sorts of puts the non-Dutch speaking people outside of our team because they cannot join the workshops [..]. It is excluding people basically, but we try to write user stories in at least full sentences so that you can easily translate it. And for user stories specifically, we write an English translation ourselves, but still not every translation is as good [and] understandable for everyone' (Analyst, Interview 3). *Also, the translation of text in diagrams is a challenge, which results in redrawing a diagram to update with the translation of the text.* 'The difficulties are mostly when also the diagrams are in English and need to be translated into Dutch [..]. You have to edit the Draw IO file and export it again and it is time-consuming, but that is the difficulty I would say' (Architect, Interview 2).

Contribution to documentation

To assess the processes in place to ensure adequate contribution to documentation. We asked the participants about the processes in place to consider the planning of documentation to ensure documentation is not forgotten due to time constraints. Also, we ask how the issues found in the documentation are reported and the support in place for external contributors.

For all four teams, *there is no process in place to ensure the planning of documentation.* A common reason for this is due to the *agile mindset of prioritising the working software and documenting less.* We first discuss the perceptions of

documentation for P1 followed by P2. 'It is difficult [to plan documentation] because we do not put so much priority on the documentation' (Analyst, Interview 4). *Due to the estimation of user stories and prioritising of tasks, there is a lack of time to focus on having quality documentation.* 'There is lack of time and what is the cause of the lack of time is, [..] that our estimates are too low and so development takes too much time, and then we do not have enough time for documentation that is one thing. Then there is pressure, [..] you need to finish a user story, and otherwise, you need to postpone delivery of a user story, like the formal delivery, because documentation is not yet done' (Architect, Interview 2). However, *if documentation is scrapped due to time constraints, nearing the end of the project, the documentation is updated.* 'that is usually the case [the documentation] is left to the end and then with two or three people we have to spend a couple of days to update and correct, that is usually how it goes, and it is a shame' (Architect, Interview 2).

Similarly, these issues are identified in the teams of P2. In P2T1, *'when there are time constraints, documentation is usually one of the earliest things that get scrapped'* (Lead Developer, Interview 7). Also, *if documentation is scrapped due to the deadline, it is difficult for the team to go back and fix it, resulting in a huge documentation debt.* 'In practice, this does not happen (fixing documentation debt) because the deadline was finished. There might be yet another deadline to work for us, so that is [..] very unfortunate, but that is the way it often goes' (Lead Developer, Interview 7). *P2T2 does not have a process in place to plan documentation.* But the assumption is that when they are nearing the end of the project, they will incorporate activities to catch up on documentation. However, there is some evidence of addressing the issue of documentation debt at the Retrospective. 'Maybe if we are finishing a project, [..]. So the only moment is not now is the is maybe the Retrospective, someone making the notes [what] could be better [with regards to documentation], but mostly, I think it is good if you have a project and after a while, you have to look back and look how you can do better, sort of an evaluation, but that is not the case at the moment though. [..] maybe it is something we can do [..] at the end of the PI or something, but [..] it is not something that is generic in the agile way of working' (Product Owner, Interview 8).

Moreover, *there is no process in place for any of the four teams to report issues found in the documentation. The teams in P2 address this via a conversation on Teams or face-to-face conversations, whereby they discuss issues about documentation.* As mentioned in Section 5.2.4 feedback or issues relating to documentation are solved immediately within P2T1, meaning the issues tend to have a high priority. Additionally, *there are no guidelines in place for external contributors*, as they hardly write or update documentation within the project. 'Usually, the team writes it' (Architect, Interview 2).

Automated documentation

We asked the participants whether any aspects of the documentation is automated. *In general, there is not so much automation of documentation* and the reasons are given as follows. *However, P1 makes use of HTML clickable prototypes and unit tests, whereas P2T1 uses C4 diagrams and unit tests, which entails some automation. P2T2 does not have any automated documentation.*

The HTML clickable prototype is an executable documentation artefact that is also used for the styling of the frontend. 'Apart from having a clickable prototype that is as much [of] a reality as it could be, we also use it to help developers using the styling components' (Analyst, Interview 3). *Automated documentation is an area that could be potentially looked into by the team.* 'We need to look into that as possible and the quality of the generated documentation. Because, I know you can generate those class diagrams, but they get quickly outdated. Usually, a class diagram that gets generated is not very readable, so you would still need to manually organise the boxes and lines' (Architect, Interview 2).

Also, *it is argued that it is difficult to automate the documentation within P2 due to the domain knowledge required.* It involves a lot of domain knowledge. 'Many aspects are a bit hard to automate. Because, it very much involves domain knowledge, which is something that cannot always be automated. [It] requires an understanding of what you want to describe, especially for very complex [functionality] it is just very complicated to do' (Lead Developer, Interview 7).

Other issues

Furthermore, we asked the participants whether there are any other challenges facing documentation that have not yet been discussed in the interviews. *A recurring issue is the time constraint.* In P2 we see that 'The main constraints are [..] the time constraints [..]. It is really related to priority constraints, [..] and perhaps also, a bit of an awareness of the importance of documentation because it always seems like something that just costs time and does not gain people anything [..]. It would be worthwhile [if] people would actually see documentation as something that is valuable in itself, that it is not all about delivering feature' (Lead Developer, Interview 7). Also, the issue of time is mentioned in the other teams such as 'Time is always an issue, documentation is never [a] priority, it is just not and it will never be, that is the main challenge' (Analyst, Interview 3).

Additionally, the Lead Developer in *P2T1 suggests that the awareness of the essence of documentation should be achieved 'through education or through discussion from the team* just to see what problems could arise from having too little documentation, and what is the best way to address this' (Lead Developer, Interview 7).

Strengths of documentation processes

Finally, we asked the participants to name what is going well with regard to the documentation at the moment. We discuss the strengths and evidence of best practices within the documentation practices in P1 and P2.

- *The user stories of P1T1 are of high quality and are constantly reviewed.* 'The user stories themselves [are going well], because we depend on them, and review [them] and everyone is involved' (Architect, Interview 2).
- *The user story template automatically adds a documentation task (P1).* 'How we maintain and make sure is that there is always a [documentation] task, so automatically if you add a user story in Azure DevOps, it will have a functional or architecture or whatever documents task with it, so you cannot escape and we can see if someone did it or not. And if it is missing, you can backtrack the person who worked on it' (Analyst, Interview 3).

- *The manager enforces the updating of documentation (P1T1).* 'What we have done now is that the manager said do not start a new user story before you finish documentation from the one you finished before. So a user story is only finished, that could be the definition of done, once every task is done. So, if there is just one small thing like documentation, it is not done we are still working on it' (Analyst, Interview 3).
- *Planning time-slots in one's own agenda to work on documentation (Analyst, P1T1).*'For me personally, [for the] functional documentation, I usually plan a couple of hours once per sprint to [update] and make sure I do the documentation by the end of every Sprint' (Analyst, Interview 3). This is also a good way to learn and improve the estimations of documentation tasks. 'That is my way of learning my task because then I have more insight. I know how much time I usually need for [minutes of meeting], so I can plan it. And then I can make sure that I have enough time to do all my tasks' (Analyst, Interview 3).
- *The availability of templates and best practices in C1M accelerate the documentation process (P1).* 'Overall, we do a really good job documenting our stuff and I think the task documentation really helps to do it more structured. And the main thing is the [SharePoint] and [C1M], I think that really helped us to focus or to at least start documentation, to know what documentation was right. Because before [C1M], we did not have these templates but let alone best practices. So, I think those best practices and templates really help you to just start. And then, it just goes on. I think the document just grows during the process' (Analyst, Interview 3).
- *The increase in awareness of documentation (P2T1).* 'The awareness of the need for documentation is becoming greater over time. So I think one year ago, we had like almost no documentation and now it is already a bit better. [..] the wiki is coming along really nicely and I think that is a very good way to put your documentation, it [is] very easy to change and very organised, [..] those are the main things that are going well' (Lead Developer, Interview 7).
- *Development of a knowledge sharing platform 'Kennisbank' (P2T2).* P2T2 started with the development of 'the Kennisbank, the knowledge side, which will which we want to expand upon' (Product Owner, Interview 8). The 'Kennisbank' is a knowledge platform being created to facilitate knowledge sharing about the solutions of the team with other stakeholders both internally within C2 and externally.

5.4 Conclusion

Throughout the Problem Investigation phase, we identified issues and strengths of the documentation within the explored projects with regard to what is written, how is it written, and what are the documentation processes within the projects. A common challenge is time constraints that result in huge documentation debt due to the Agile mindset of focusing more on implementing and testing and documenting less. All teams use the Refinement session to clarify requirements. While C1M introduces templates and best practices for writing documentation, the downfall is that templates such as that for a user story tend to get outdated easily. Even though P1 defines documentation tasks and P2 does not, the scope of the documentation is not clear for either project.

Moreover, P2T1 faces the challenge of finding the right balance between docu-

menting and making the code self-documented. The lack of a clear file structure is seen in P2T2 which hinders the process of writing documentation. In P1, the update and completion of the documentation solely rely on the assignee rather than a shared team responsibility. The larger the documentation artefact, as seen in P1, the more difficult it is to maintain it. Hence, P2T1 approaches this issue by making the documentation more modular. For all teams but P2T2 feedback and review from external stakeholders are lacking. P2T2 is addressing the feedback from stakeholders by realising a knowledge-sharing platform. A common pattern of feedback among the teams is through the four-eyes principle. Lastly, the processes of planning documentation, encouraging feedback and incorporating feedback are absent in all of the teams.

Chapter 6

Treatment Design

We consider the problems identified in Chapter 5 such as time constraints, lack of awareness, lack of a clear documentation scope, planning, review, and estimation, among others as areas of potential improvement. With these insights, we propose a treatment to address the challenges found in documentation as the second phase of the Design Science cycle. Our approach is based on a combination of a semi-systematic literature study with the guidelines of the treatment design by Wieringa (2014). For the literature study, we searched for literature within the past decade on software process improvement within agile teams. The guidelines for documentation practices in agile development (Section 2.5.5) are rather abstract to be applied in practice. To the best of our knowledge, there is no framework or Software Process Initiative that addresses the challenges facing documentation in agile teams. Nonetheless, based on the research of Hess et al. (2019), we considered the recommendations to develop a checklist as a guideline for face-to-face meetings [108]. Then, we researched further into the potential of software process improvement via informal communication. In the literature, we found the benefits of team meetings and the recommendations for a game-based and data-driven approach to the Retrospective. With these insights, we design the Documentation Improvement Framework via Feedback Cycles (DIFFC).

The remainder of this chapter is as follows. We explain the reasons why one treatment framework does not fit all large-scale agile projects in Section 6.1. Then, we address the state of Software Process Improvement initiatives from the literature in Section 6.2. Subsequently, we discuss the potential for process improvement in self-organising agile teams using informal interactions by means of feedback cycles in Section 6.3, as well as the challenges faced during these feedback cycles such as Daily Scrum, Retrospective, and Sprint Planning meetings in Section 6.4. We zoom in on the facade of our treatment based on informal interactions. In parallel, we identify requirements for the treatment design in the aforementioned sections, which are italicised and assigned as labels R1-R6. Then, we propose the DIFFC method in Section 6.5 in an attempt to address the issues found in the documentation with both practical and scientific contributions. Finally, we wrap up in Section 6.6.

6.1 Variations in Scaled ASD Projects

We discuss the variations among scaled agile projects and why 'one size' of the treatment does not fit all contexts. Firstly, there is no one size fits all solution for approaching agile software projects [7], and this applies to improving documentation practices in scaled ASD projects, as the documentation needs of each project are different. Our sixth sub-research question covers the design of a framework to improve documentation practices in scaled agile software projects. However, with the insights from the case studies, we see variations in the context and structures of these projects. This also confirms the statement in project management literature that *'one size does not fit all'* [109, p. 14]. Moreover, in a study conducted by Shenhar et al. (2002) on the differences among projects and classifying projects according to frameworks, the authors conclude that *'one (project classification) framework does not fit all either'* [109, p. 14]. In the context of documentation practices in scaled agile development, we highlight the variations.

The rationale for use and the quantity of documentation differs per project. In P1, it was evident that the number of documentation artefacts used is higher than that of P2. The goals of the projects to deliver a product to a client or for internal use varies. The organisation and processes within P1 are driven by the internally created method of C1 (C1M), whereas the P2 is organised and structured by an adaptation of SAFe within C2. C1M requires more documentation artefacts from the teams in the form of formal documents whereas the documentation in P2 is mainly informal documentation artefacts. Having external clients as stakeholders also influences the governance, quality assurance, and external follow-up rationales. For instance, in P1T1, a Maintenance Guide is used to accompany a release and that should be approved by the customer.

Moreover, the process maturity level differs per team. C2 adopted SAFe about half a year ago and P2 is the only team as yet that uses SAFe within the company. On the other hand, C1 adopts C1M for almost three years, and almost all the projects within C1 adopt the methodology. Therefore, there is more experience and expertise in the agile method practices of P1 compared to P2. In P2, the process maturity level even differs per team. P2T1 for instance has worked with Scrum for about 4 years, whereas P2T2 is relatively new to the agile way of working, and adopted agile practices after SAFe was introduced. With these findings, we conducted further research into Software Process Improvement (SPI) and the Capability Maturity Model (CMM). CMM was designed to aid software organisations in selecting process improvement strategies based on their current process maturity [110]. The model distinguishes five levels of process maturity in software projects from 'initial', where the software process is categorised as "ad hoc, and occasionally even chaotic' to 'optimising', where there is continuous improvement to the software processes by means of feedback and innovation [110].

6.2 Software Process Improvement (SPI) in ASD

SPI refers to activities that are aimed at assessing and improving the processes and practices involved in software development [111]. It has roots in plan-driven

methods in software development, however, over the past two decades, it is increasingly being adopted in ASD [112]. In a systematic literature study on SPI in ASD by Santana et al. (2015), two fundamental approaches to SPI in agile were identified [112]. The approaches were based on the dissertation of Salo (2007), where the top-down and bottom-up approaches are discussed [113]. The top-down approach focuses on the utilisation of organisational procedures and explicit knowledge transfer [113]. This approach is based on a plan-driven approach to software processes whereas the bottom-up approach is more inclined toward the agile mindset. The bottom-up approach focuses on the adaptation of processes to the "contextual needs of project teams and improving the contextual needs of individual project teams" [112, p. 329]. It distinguishes two sub-approaches:

- "Adapting the process to the contextual needs of individual project teams" [113, p. 59].
- "Improving the effectiveness of individual project teams" [113, p. 59].

These bottom-up approaches are the basis for Agile SPI which reflects the first principle of the agile manifesto [112], where individuals and interactions are prioritised over processes and tools [5]. As self-organising teams as referred to in ASD, they 'are usually responsible for managing and monitoring their own processes and executing tasks' [114, p. 27]. Hence, *the adoption of lightweight process improvement is more suitable for this context, rather than introducing a heavyweight framework for process improvement* **(R1)**.

A treatment framework that requires a process-heavy approach by introducing new processes to the already existing processes and activities of the various scaled agile methods is not preferred due to the potential risk of low adaptability. ASD recommends a more 'adaptive' and 'exploratory' approach towards its processes, with eliminating unnecessary formalities [115]. It discourages plan-driven approaches that are process-heavy and entail high ceremony. An example of the high ceremony mindset can be seen as 'I write a document because now is the prescribed time to write that document even though nobody can say why that document is necessary or valuable' [115, p. 57]. In essence, Agile Software Development is lightweight, allowing organisations to adopt its practices to suit their context. Also, the methods used by large-scale software projects such as SAFe, DAD, LeSS, and internally created methods, introduce additional processes and activities making it more difficult to introduce a heavyweight process improvement. Implementing heavyweight approaches demand extensive assessment activities and are costly to adopt [116].

Besides, as self-organising Agile teams, a big part of adaptability change is working in iterations and constantly improving based on feedback among the team members [48]. In Section 2.2.4, we discussed the feedback cycles in ASD, specifically in Scrum. In these meetings, artefacts may be used such as the Sprint Backlog in the case of the Daily Scrum, as evident in P1 and P2. However, the main process improvement feedback is elicitation through verbal communication [48]. *With verbal communication in an offline, online, or hybrid meeting setting, the team members discuss progress and problems, but also share suggestions for improving processes within the team* **(R2)**. Therefore, **the treatment design focuses on a lightweight approach to improve documentation**

practices from within the team in a large-scale ASD project.

6.3 Process Improvement via Informal Interactions

With scaled agile methods focusing more on informal communication rather than extensive knowledge transfer [10], there are still a considerable amount of artefacts in the explored cases. Informal communication is one of the fundamental factors for success within ASD [117]. "Communication and collaboration are at the heart of agile software development" [118, p. 298]. However, heavily relying on informal communication introduces the risk of knowledge evaporation, especially in the case of larger software projects [117]. We argue that informal communication among the team members at feedback sessions can contribute to improving the documentation process.

In Section 6.1, we introduced the idea of improving documentation practices with a lightweight approach which is based on the feedback sessions among the team. The most popular agile methodology, Scrum, evolves around four feedback cycles which are the Daily Scrum, Retrospective, Sprint Planning, and Sprint Review [115]. Since the Sprint Review is mainly for evaluation purposes and may involve external stakeholders, we scope the treatment design to *focus on the internal meetings that may be used to coordinate processes within the team* **(R3)**. The reason for this is to investigate the effectiveness of the process improvement initiative within the self-organising team. Therefore, we discuss the Daily Scrum, Retrospective, and Sprint Planning. ASD involves self-organising teams who rely on daily meetings to coordinate their work and make decisions [119]. The Daily Scrum is a meeting of approximately 15 minutes whereby the team members are kept up-to-date about the progress of the Sprint and discuss any issues or impediments [120]. The Retrospective is seen as an 'inspect-and-adapt' activity that occurs at the end of the sprint with the focus on continuous process improvement [115]. This activity facilitates discussions about the processes within the team, as well as the results of the iteration and possible improvements to the processes for the next iteration(s) [121]. Sprint Planning initiates the next sprint. During this meeting, estimates are made for the user stories on the Sprint Backlog for the next iteration [115].

In a study conducted by Stray et al. (2012) to investigate team meetings in ASD, the authors discuss the relevance of team meetings. Team meetings are essential for the coordination of team members, tasks, and tools [119]. In turn, a software team integrates its members' efforts, as well as their activities and goals, with suppliers, clients, and other organisations both inside and outside the firm through these meetings [119]. Additionally, through team meetings, the team is able to share information, combine expertise and address problems, and make decisions [119]. During a week, a software developer usually spends several hours in various types of meetings such as planning, review, retrospective, daily meetings, and customer meetings [119]. Also, team performance is strongly linked to the effectiveness of teamwork coordination [122]. Hence, team meetings with a more proactive approach such as problem-solving interaction and action planning contribute to higher team productivity [123].

The benefit of incorporating process improvement into these feedback meetings is that these meetings already exist and are used by the project teams. For example, all the teams in P1 and P2 organise Daily Scrum, Retrospective, and Sprint Planning meetings. In the literature, the Daily Scrum is perceived as the most essential medium of communication among the team in Scrum [124]. Considering the relevance of these channels of communication, using the Daily Scrum, Retrospective, and Sprint Planning as a platform to improve documentation practices provide the benefit of reusing existing processes without introducing any additional complexity to the preexisting activities of the team, making it a more adaptable approach. Therefore, we incorporate the use of preexisting feedback cycles within agile teams into our treatment design.

6.4 The Challenges Associated with Feedback Cycles

Despite the benefits of using feedback cycles as a medium for process improvement, there are some hindrances. In a study conducted by Hess et al. (2019), to address the challenges facing requirements' communication in agile teams, two guidelines were recommended. Even though the guidelines are directed towards addressing the gap of details in requirements in ASD, the second guideline is rather generic and may be applied in other contexts. This guideline proposes the introduction of a checklist to guide feedback meetings such as "sprint meetings in order to assure that important information for agile team members is explicitly discussed and clarified" [108, p. 9]. The daily scrum meeting is centred around at least these three main questions, 'What did I do yesterday?', 'What will I do today?', and 'Do I see any impediments?' [120]. In contrast, the Retrospectives do not have a prescribed set of steps to support the discussions of estimation and process improvement, among others [121]. Retrospectives games are adopted during Retrospectives which are a set of interactive group activities to keep the discussions interesting and encourage active participation in those discussions [121]. *Having a guide for the structure of these meetings will therefore be an interesting prospect towards process improvement of documentation practices within the agile team* **(R4)**.

Another problem facing Retrospectives is the lack of data. Even though Retrospective games keep these meetings interesting and encourage involvement, a critique is that the 'hard facts' are ignored [121]. It is without a doubt that these interactions are highly relevant to improving team satisfaction [125]. Resulting in a list of actionable items as a potential for process improvement within the team [126]. However, Matthies (2020) emphasises the relevance of modern software project data and proposes the need to combine insights from the data with the informal interactions [121]. Therefore, we incorporate this proposal within our treatment design. *The framework should combine a data-driven approach with informal communication* **(R5)**. Software project data is very essential in providing insights into the progress and the issues facing the developed product [127], which also includes the documentation artefacts.

The integration of tools in ASD provides access to agile teams about the project status without much effort required to collect and prepare this data [121]. Cat-

egories of tools such as version control systems, issue trackers, software testing suites, status monitors, and communication tools provide the team with information about the progress and quality of the various aspects of the projects ranging from code quality, frequency of code commits, to the frequency of communication, among others [121]. This data is certainly useful to guide the discussions on processes and results within the team rather than having a subjective discussion about process improvement only based on 'feelings' [121]. In justifying the proposal to incorporate guidelines to the structure of retrospectives, a recommendation to start with the facts, a 'set the stage' phase was introduced [121]. This phase as the start of the Retrospective allows the use of data to provide more insights into the progress and behaviour of the team [121]. Matthies (2020) presented a quote by Derby and Larsen (2006), that supports the aforementioned, "Without a shared picture, people are working from a narrow set of data - their own" [128]. Retrospectives should start with data of the team's milestones of the last iteration [121]. However, in the context of documentation activities, the question thus remains whether these data provide insights into the progress and quality of documentation within the project.

Metrics are an important aspect of software process improvement. Pandian (2003) explains that the 'ability to improve is aided by the ability to measure' [129, p. 25]. Metrics support decision-making and provide data for identifying issues which eventually results in problem-solving [129]. A number of the artefacts and tools used within ASD provide insights into the progress of the product and artefacts. User stories are tracked in issue tracking systems, whereby burn-down chats are used to visualise the velocity against the planned effort [91]. Velocity is measured in story points which indicates the team's productivity through the amount of work accomplished by the team within a sprint [130]. The higher the velocity, the higher the number of sprints required to complete the defined story points [130]. Additionally, the number of requirements, defects, and customer satisfaction scores are performance indicators adopted by Agile teams [131]. Therefore, *if documentation is made measurable like other tasks within the user story, then it should provide the opportunity of monitoring the progress of documentation* **(R6)**. Data on how much effort was spent for a documentation task in comparison to the planned effort, contributes to better estimation. As a result, the risk of documentation being scrapped, forgotten or moved to the end due to time pressure, is reduced with better planning and estimation.

6.5 Documentation Improvement Framework via Feedback Cycles (DIFFC)

In this section, we propose Documentation Improvement Framework via Feedback Cycles (DIFFC) to address the issues found in documentation activities in large-scale ASD projects. The focus is on process improvement on a team level via informal discussions. DIFFC is a lightweight framework that can be incorporated into the existing feedback cycles of the agile team. The rationale behind this framework is to stimulate process improvement actions to address the issues with documentation within the team itself, rather than imposing external processes and guidelines for a self-organising agile team. The framework

is not to be rigidly implemented, instead, it may be used as a guide to incorporate structure already existing feedback cycles such as the Daily Scrum, Retrospective, and Sprint Planning, among others.

Our treatment addresses the following requirements:

- R1: The framework should be lightweight and easy to adopt by agile teams. (Section 6.2)
- R2: The framework should incorporate the use of verbal communication, whereby the team members discuss progress and problems, but also share suggestions for improving processes within the team. (Section 6.2)
- R3: The framework should focus on the use of internal meetings to coordinate processes within the team. (Section 6.3)
- R4: By applying the framework, a Retrospective should be accompanied by guidelines for its structure. (Section 6.4)
- R5: The framework should combine a data-driven approach with informal communication. (Section 6.4)
- R6: Documentation should be made measurable to provide the opportunity of monitoring the progress of documentation activities. (Section 6.4)

As outlined in Section 1.3, our research aims at both scientific and practical implications. Therefore we introduce the DIFFC model for practitioners in Section 6.5.1 and the scientific overview of the framework in Section 6.5.2.

6.5.1 DIFFC for Practitioners

To ensure the DIFFC is adopted by practitioners, we provide a simplified and comprehensive model of our framework. Here, the activities within the phases are more compressed making it more adaptable for multi-team software projects with pre-existing structures and processes in place. In Figure 6.1, we present the model of DIFFC for practitioners. The Planning phase entails activities to ensure that the user stories include documentation tasks that are estimated and assigned. This should ensure that documentation tasks are included in the sprint estimation and are not forgotten. On a daily or frequent basis, the progress of not only the user stories but also the documentation tasks, as well as the process improvement actions (derived from the Reflecting phase), should be reported during the Monitoring phase. Also, if any dependencies are hindering the progress of both kinds of tasks, that may be discussed during the progress update meeting. The purpose of this phase is to ensure that the team is aware of the progress of the various tasks.

Moreover, the most important phase of this model is the Reflecting phase. In order to make this phase successful, an agenda should be made with regard to tackling specific issues facing documentation processes and artefacts within the team. The activities of the Reflecting phase are divided into two parts, the first part combines the perceptions of the team members with a data-driven approach to team reflection. For instance, during a Retrospective, the team members may begin by sharing their perceptions about the results and processes within the previous sprint. Also, the progress of the process improvement tasks from the previous iteration is evaluated. Then, insights from Software Project

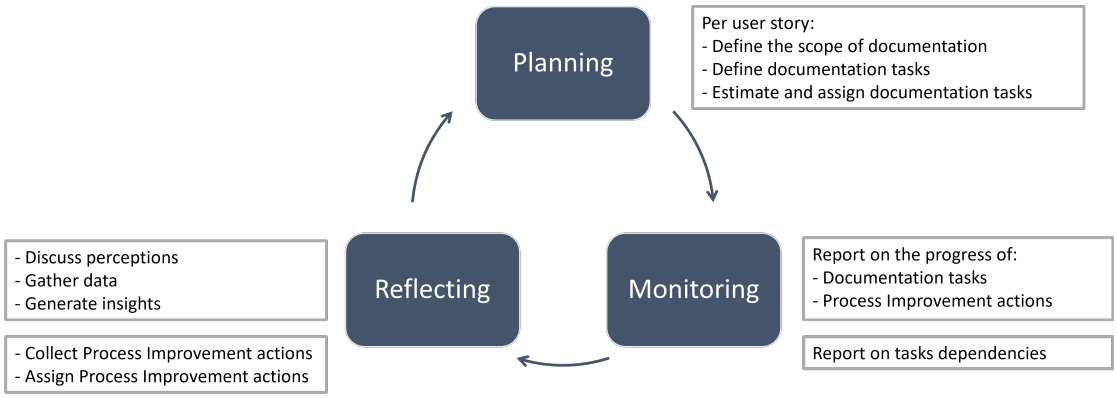


Figure 6.1: An overview of the DIFFC framework aimed at practitioners.

Data such as the burn-down chart, progress of deliverables, release information, documentation debt, etc. are gathered based on the agenda of the meeting and presented to the team. Based on the data, the team can begin to elicit ideas on how to address the problem. To support an interactive and collaborative approach, we recommend the use of Retrospective games in Section 6.5.3. Also, it is important to incorporate the role of the facilitator, if not already present in Retrospectives. From the literature, it is recommended to have the facilitator support the activities of the Retrospective and refrain from making decisions to allow active participation of the team members [132]. Using the game-based approach, ideas are generated by the team and process improvement tasks are agreed upon. These tasks are then assigned to team members as responsible for the execution of the tasks.

As mentioned in Section 6.5, the goal of the DIFFC model is not to have all activities incorporated within the process of the team, but rather to evaluate one's existing processes and incorporate the recommendations of the model. Having discussed the variations of scaled ASD projects in Section 6.1, the expectation for this model is that not all the proposed activities might be applicable to a certain case. Therefore, this framework may be used as a guideline for practitioners, if an activity or sub-activity already exists in the current process, it may be skipped when incorporating it into the process of the team. For instance, during the case exploration, we identified that the teams in P1 already have documentation tasks which are assigned to a responsible, but not a reviewer as yet, and the tasks were not estimated. In the case of P2, documentation tasks are not defined as part of the user story. In the case of monitoring, both projects report on the progress of the user stories during the Daily Scrum, but there is no specific mention of documentation tasks. For reflection, the meetings follow an informal structure during the Retrospectives whereby the perceptions of the team members are discussed and eventually, some process improvement tasks are made. Hence, the structure and activities (except the discussion of perceptions) we introduce in the Reflecting phase are not yet implemented by these projects. Hence, we test the DIFFC model in the explored cases, to investigate whether our model addresses the problems facing documentation in large agile software projects, and how it contributes to software process improvement.

6.5.2 DIFFC for Science

In Figure 6.2, we adopt the Process Deliverable Diagram (PDD) notation [133] to describe the main processes and activities as well as deliverables in the DIFFC method. The method consists of three phases. The first phase defines the Planning phase where DOCUMENTATION TASK(s) are explicitly added to user stories to ensure that the documentation aspect of the USER STORY is made measurable and not forgotten. Here, the need for documentation within the USER STORY is discussed among the AGILE TEAM at a feedback cycle such as the Sprint Planning. The REASON FOR DOCUMENTATION determines the scope of the documentation. The SCOPE OF DOCUMENTATION is used to address the question of *'what is enough documentation?'*. With these insights, the AGILE TEAM defines a DOCUMENTATION TASK as part of the USER STORY. This task is then estimated by the AGILE TEAM, and assigned to at least one RESPONSIBLE and one REVIEWER.

Subsequently, the second phase defines the monitoring of both the DOCUMENTATION TASK(s) and IMPROVEMENT TASK(s). The RESPONSIBLE assigned to a task gives a PROGRESS UPDATE on the completion, as well as any issues or dependencies on other tasks. The monitoring phase can be seen as a progress update to keep the AGILE TEAM in sync with the progress of the various tasks. This phase can be incorporated into the feedback cycle focused on progress updates such as the Daily Scrum.

The third and most important phase is the Reflecting phase. This is described as such because it introduces a new change that was not evident in any of the explored cases. In both P1 and P2, the Retrospective meeting is used to discuss the perceptions of the team members of the previous sprint and propose process improvement ideas. However, there were no formal guidelines for this meeting, the topic of documentation is barely discussed, and these meetings did not make use of any project data. With DIFFC, we propose a data-driven approach to team reflection, that combines the perceptions of the TEAM MEMBERs with SOFTWARE PROJECT DATA. The Reflecting phase can be incorporated into the feedback cycle focused on progress updates such as the Retrospective in Scrum. The first three activities of this phase were adopted from the book of Derby and Larsen (2006) where the authors proposed a structure for Retrospectives by making use of Software Project Data (SPD).

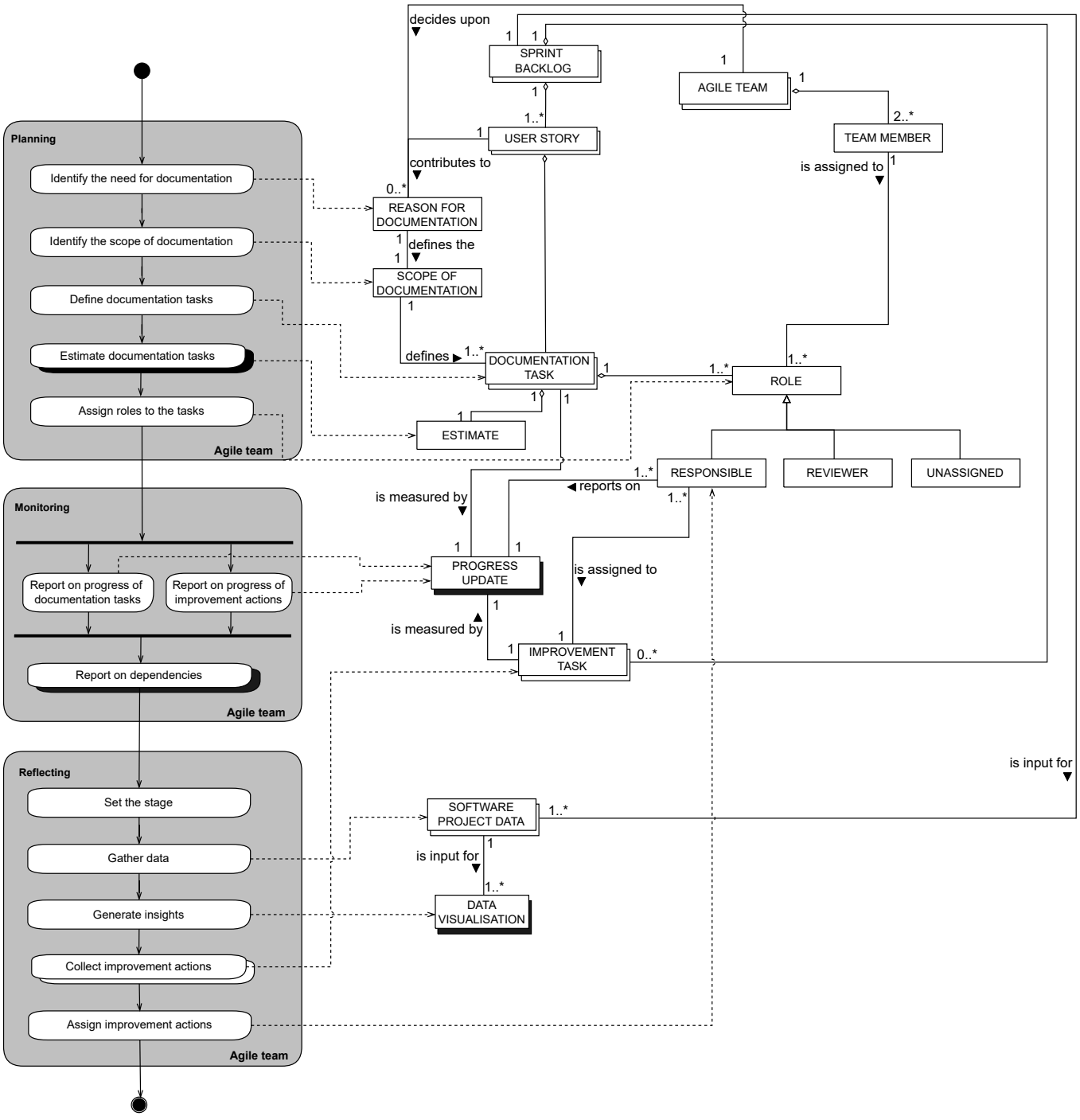


Figure 6.2: A process deliverable diagram of the proposed DIFFC method.

Derby and Larsen (2006) identified proposed five stages for the structure of a Retrospective meeting, known as 'set the stage', 'gather data', 'generate insights', 'decide what to do', and 'close' the Retrospective [128]. Setting the stage prepares the team for the activities of the Retrospective. This phase entails reviewing the goal of the meeting, reviewing the agenda, checking in, and evaluating working agreements made in the previous Retrospective [128]. Afterwards, during the gathering data phase, a shared picture is introduced with the data of what happened during the previous iteration [128]. 'Without data, the team is speculating on what changes and improvements to make' [128, p. 50]. Subsequently, generating insights allows the team to evaluate their data and draw meaningful conclusions from it. These activities help the team interpret the data, analyse it, and discover implications for change [128].

The team gathers SOFTWARE PROJECT DATA to generate insights into variables such as the number of USER STORIES completed, the number of DOCUMENTATION TASKS successfully completed, and the actual effort of documentation versus the planned effort, among others. With these insights, not only can the AGILE TEAM focus on eliciting ideas for process improvement, but for example, they may make decisions on whether the cost of updating documentation is in line with the effort spent on the actual implementation of the user story. After the SOFTWARE PROJECT DATA has been visualised, the next activities are to collect and assign process improvement actions.

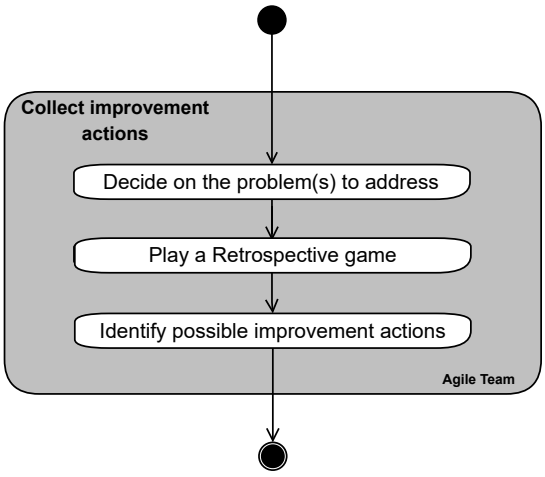


Figure 6.3: A PDD showing the sub-activities for the 'Collect improvement actions' activity.

We expand upon this research with the work of Mesquida et al. (2019) on developing a collaborative game toolbox for SPI in agile teams [132]. The authors introduced a 'Playing Up' step during Retrospectives. The purpose of this activity is to explore old processes in a new way [132]. We incorporate the 'Decide what to do' activity [128] with the 'Playing Up' activity to combine a data-driven approach with an interactive approach, to stimulate the team to interactively participate in generating ideas on addressing issues identified from the project data. The idea to include incorporate a data-driven approach with the game thinking approach was inspired by the proposal of Matthies (2020)

to use SPD in Retrospectives [121]. Hence, the 'collect improvement actions' activity consists of three sub-activities as shown in Figure 6.3. This entails selecting a problem to address, selecting and playing a Retrospective game, and defining a list of IMPROVEMENT TASKs to address the problem in question. An IMPROVEMENT TASK is assigned to at least one RESPONSIBLE and it is added to the SPRINT BACKLOG of the next iteration.

6.5.3 Retrospective Games

In the previous section, we briefly touched upon the relevance of Retrospective games. As an essential activity of the DIFFC method, we elaborate more on the game-based approach to Retrospective based on the literature.

Agile teams mainly trigger process improvement initiatives during the Retrospective meetings, this feedback cycle takes place at the end of the sprint with the focus on continuous process improvement [134]. An essential component of these feedback cycles is the tools and techniques used. The game thinking approach towards SPI via Retrospectives is discussed in the literature including the work of Jovanović et al. (2016). The authors emphasise that the use of games in Retrospectives increases team motivation and contributes to a positive impact on social behaviour and team development [126]. Therefore, incorporating this method in the software development process has a good impact on team members' motivation and encourages the use of task-oriented resources [126].

Before selecting a Retrospective game, Mesquida et al. (2019) recommend that the Retrospective meeting should be planned and prepared in advance by the project team [132]. The agenda for the Retrospective meeting should be decided upon by considering social and environmental factors [132]. These factors include 'group dynamics, maturity, social and interpersonal aspects' of both the team and the individual team members [132, p. 107]. Especially for group dynamics, Mesquida et al. (2019) pinpointed the group development stages by Tuckman (1965), which are the forming, storming, norming, and performing stages [135].

- *Forming* is the first stage of group dynamics where the team members are observing and orienting themselves toward one another, and adopting the expected and recommended working methods [135].
- *Storming* is the second stage of group development, during which members of the group may frequently experience disagreements in the course of their regular activities [135].
- *Norming* is the third phase and it is assumed that the disagreements and various working styles are aligned into one recognised way of working and interacting, group members communicate more, they have a feeling of belonging to the team, and they begin to be more proactive in terms of improvement [135].
- *Performing* is the last and optimal phase, where the group is working most effectively at this stage, individual tasks are recognised as group assignments, everyone is focused on completing the task as a group rather than individually, roles are interchangeable, and the overall task is completed successfully [135].

Another factor to consider when selecting a Retrospective game is 'the meeting time phase in which the game is most applicable' [132, p. 108]. By applying the

principles of design thinking, Mesquida et al. (2019) propose three time phases for a meeting known as 'Warming up', 'Playing up' and 'Wrapping up' [132]. The first phase is aimed at stimulating ideas. At this time of the meeting, it is recommended to propose innovative ideas and opportunities during this phase but not to engage in critical analysis or scepticism' [132]. The playing up meeting time phase is an exploratory phase appropriate to explore old processes in a new way [132]. Finally, the wrapping up phase should include a critical and realistic overview, which entails a conclusion regarding decisions, and a list of activities that need to be completed before the following meeting (the next Retrospective) [132].

In addition to the aforementioned factors, the setup of Retrospective meetings is essential. In order to allow all team members to participate and have a say in the decision process, it is crucial that the facilitator refrains from making decisions [132]. To promote collaboration and encourage idea generation, Mesquida et al. (2019) combined the game-based approach with design thinking and introduced a collaborative game toolbox that 'contains a set of games to be used by the team members during a meeting with the main objective of improving communication, cohesion, and coordination' [132, p. 108]. The toolbox consists of twelve games which are summarised as follows.

- *Game 1: Defining the team principles* [132, p. 108]. During this activity, participants are encouraged to converse and engage in debate. This game's objective is to democratically establish the team's core values. These guidelines should outline the qualities that team members should encourage or refrain from [132].
- *Game 2: Future Facebook posts* [132, p. 108]. With this activity, team members are encouraged to focus on their desired future course. The aim of this exercise is to prioritise and come to an agreement on the team's short-, medium-, and/or long-term goals [132].
- *Game 3: Lessons learned* [132, p. 108]. It alludes to looking for lessons following a significant event. The objective of this game is to compile project lessons that may be applied to future projects [132].
- *Game 4: Peaks and valleys timeline* [132, p. 108]. It is a useful method for identifying team interactions as well as "ups and downs." The objectives of this activity are to recognise the issues which were apparent during a previous working period and search for potential solutions to ensure that they do not occur again in the future [132].
- *Game 5: Role expectations matrix* [132, p. 108]. In order to prevent future conflicts brought on by unspoken or concealed expectations, this activity seeks to understand how each member might profit from the others. This game's objective is to clarify and establish team members' expectations for each of the defined roles [132].
- *Game 6: Roles we play* [132, p. 108]. This activity facilitates a conversation about all the roles people play in life. It is beneficial for a group that starts working together in particular. The objectives of this exercise are to learn about team members' interests, preferences, and hobbies as well as to build and reinforce the bonds between coworkers by fostering new connections and relationships [132].
- *Game 7: Speed car* [132, p. 108]. This activity is a simple approach for assisting the team in determining what causes it to move more quickly and what

causes it to move more slowly. The objectives of this game are to uncover team productivity-affecting strengths and weaknesses, potential future dangers, and risk mitigation strategies [132].

- *Game 8: Starfish* [132, p. 108]. It involves obtaining information and assisting team members in comprehending the perceived value of one another. This activity's objective is to evaluate the team's tasks so that we can decide which ones to prioritise and which ones to drop [132].
- *Game 9: The team is – is not – does – does not* [132, p. 108]. By describing the team, the exercise highlights both the team's strengths and weaknesses. In this game, everyone must agree on the characteristics that define the team [132].
- *Game 10: Understanding the group knowledge* [132, p. 108]. The objectives of this activity are to evaluate a team's knowledge and abilities and to suggest ways to improve team knowledge. The group knowledge is evaluated by responding to the following statements: 'we know that we know', 'we know that we don't know', 'we didn't know that we know', and 'we didn't know that we don't know' [132].
- *Game 11: Visual phone* [132, p. 108]. It is a motivator for better communication and improving interpretation. The purpose of this game is to create a bond between team members and give them the confidence to initiate a meeting [132].
- *Game 12: Who-what-when steps to action* [132, p. 108]. This activity helps to define obligations among team members by clearly stating what each team member is expected to perform and by when. The objectives of this activity are to identify and plan the work that will be completed over the next period and to make the tasks completed by other team members visible [132].

To accompany the descriptions of the games, Mesquida et al. (2019) developed a platform known as FunPMBOX[1]. According to the website, it emerged from a collaborative project at the University of the Balearic Islands. This platform allows researchers and practitioners to find instructions about the various games. For each game, a recap of the goal, instructions, duration, number of players, group stage, meeting time phase, and a list of materials needed to play the game is presented. However, currently, agile practitioners mostly rely on the web and other sources for knowledge to assist their daily work as opposed to the literature [136]. Retromat[2] is an online tool used to plan Retrospective agendas [121]. The tool provides a wide range of activities for the different phases of Retrospective meetings [121]. We include this as an alternative for practitioners who may prefer to use these activities instead of the game toolbox proposed by Mesquida et al. (2019).

6.6 Conclusion

The problems identified in the Problem Investigation phase of this research were an inspiration for the treatment design. We started by looking at the variations of scaled ASD projects such as the adopted method, the process maturity level and the structure and process in place. These variations pose a challenge to propose an extensive treatment. Therefore we looked at the benefits of the pre-existing structures and processes in place within agile teams. Informal

[1] https://funpmbox.github.io/
[2] https://retromat.org/en/?id=43-128-69-117-53

communication by means of feedback cycles is a strong contributor to addressing issues and improving processes within these teams. Hence, we dived deeper into the literature about informal communication, feedback cycles and SPIs. The recommendations of Hess et al. (2019) were a starting point with regard to incorporating guidelines in team meetings. We proposed the DIFFC model, a lightweight and easily adaptable framework, that utilises the preexisting forms of feedback cycles within agile teams. The DIFFC model ensures the planning, scope definition, estimation and progress report of documentation within user stories. As well as incorporating a data-driven and game-based approach with the preexisting discussions of perceptions as a process improvement initiative. Lastly, we recommend twelve games from the literature for the game-based approach.

Chapter 7

Treatment Validation

We implement the third and final phase of the design science cycle. We validate the proposed DIFFC model by applying it to a multi-team ASD project. "The goal of validation research is to develop a design theory of an artefact in a context that allows us to predict what would happen if the artefact were transferred to its intended problem context" [21, p. 59]. We adopt the guidelines of a single-case mechanism [21] experimental approach for our treatment validation. This type of experiment is used to test a mechanism in a single object of study and may be used in validation research to test treatment models [21]. However, our validation research is centred on two cases, the first is explored longitudinally and the second is studied preliminary due to resource constraints. In Section 3.1, we adopted the GQM approach to describe the overall goal of this research. Hence, we use the GQM approach again to describe the goal of this validation research. The goal of our validation research is to:
Analyse *the effectiveness of the DIFFC model,*
for the purpose *of evaluating its effectiveness*
with respect *to reducing documentation debt*
from the point of view *of agile project teams*
in the context *of multi-team projects.*

Our goal is to investigate how the DIFFC model addresses the issues found in the documentation and how it contributes to the reduction of documentation debt within large-scale agile software teams. The remainder of this chapter describes the experiment as well as the evaluation.

7.1 Research Approach

In the literature, research on SPI mainly focuses on software development outcomes. To the best of our knowledge, most papers that study the impact of SPI initiatives on software teams, present analysis from the perspective of software project data and not so much on the observation of adoption in practice. According to Unterkalmsteiner et al. (2011), Pre-Post Comparison is the most used approach to evaluate the effectiveness of SPI initiatives [137]. Using backward sampling to study the cited works by the author, which used Pre-Post

comparison we found that the evaluation of the Pre-Post comparison approach is based on the context of the research itself, and there was no model used that can be reused in this research.

Therefore, our validation research covers two dimensions. We adopt the Pre-Post Comparison approach [137] to measure the effectiveness of the DIFFC model with regard to the Software Project Data (SPD) on documentation. We compare the SPD on documentation tasks of the sprint before the model's introduction (Sprint n) with the results of the sprint(s) after the model's introduction. We measure the following variables:

- V1: Number of planned documentation tasks on the sprint backlog.
- V2: Estimated effort for documentation in hours per sprint.
- V3: Number of completed documentation tasks per sprint.
- V4: Completed effort for documentation in hours per sprint.
- V5: Number of documentation tasks in the project backlog.

With these insights, we can determine whether the DIFFC model enhanced the documentation process within the team by completing planned documentation tasks within the estimated time and whether it contributes to a decrease in documentation debt.

In parallel with the Pre-Post Comparison, we observe the feedback cycles of the teams whereby the phases of the DIFFC model are applied to the Sprint Planning, Daily Scrum, and Retrospective. These are described in detail in the subsequent sections, Sections 7.2 and 7.3. We adopt the research model of Green et al. (2005), as shown in Figure 7.1, where the authors studied the impacts of quality and productivity perceptions on the use of software process improvement innovations [101]. The authors also stated the following hypotheses:

- H1: Higher perceptions of software *quality* improvements from using a SPI will be associated with higher perceptions of the *usefulness* of the SPI [101, p. 545].
- H2: Higher perceptions of developer *productivity* gains from using a SPI will be associated with higher perceptions of the *usefulness* of the SPI [101, p. 545].
- H3: Higher perceptions of the *ease of use* of the SPI will be associated with higher perceptions of SPI *usefulness* [101, p. 545].
- H4: Higher perceptions of the *usefulness* of the SPI will be associated with higher levels of SPI *use* [101, p. 545].
- H5: Higher perceptions of the *ease of use* of the SPI will be associated with higher levels of SPI *use* [101, p. 546].

Even though the research of Green et al. (2005) tackles SPI from a development perspective, we adopt their approach and tailor it to measure the impact of the DIFFC model on addressing challenges in documentation. Our hypotheses are therefore formulated based on the aforementioned hypothesis on the impact of SPI on software development by Green et al. (2005).

- H1: Higher perceptions of documentation *quality* improvements from using the DIFFC model will be associated with higher perceptions of the *usefulness* of the DIFFC model.

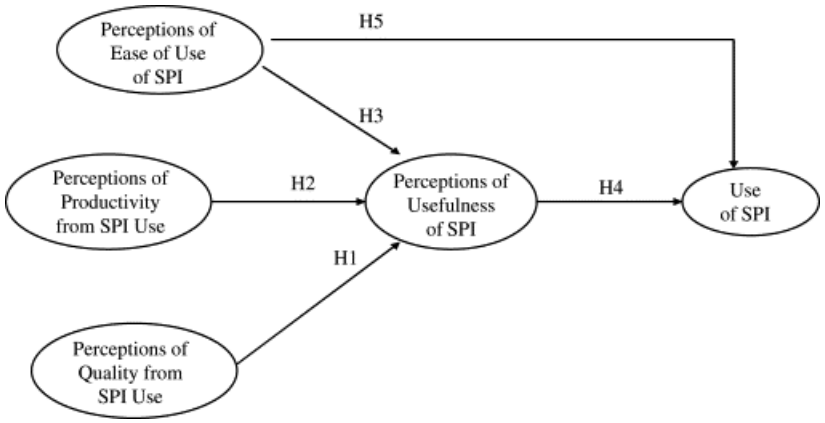


Figure 7.1: An overview of the research model adopted from [101, p. 546].

- H2: Higher perceptions of documentation *productivity* gains from using the DIFFC model will be associated with higher perceptions of the *usefulness* of the DIFFC model.
- H3: Higher perceptions of the *ease of use* of the DIFFC model will be associated with higher perceptions of the *usefulness* of the DIFFC model.
- H4: Higher perceptions of the *usefulness* of the DIFFC model will be associated with higher levels of the *use* of the DIFFC model.
- H5: Higher perceptions of the *ease of use* of the DIFFC model will be associated with higher levels of the *use* of the DIFFC model.

We explain the variables: *ease of use*, *productivity*, *quality*, *usefulness*, and *use*. The *perceived ease of use* defines the complexity of the process improvement initiative whereas the *perceived usefulness* defines the relative advantage of adopting the process improvement initiative [138]. Davis (1989), introduced the variables ease of use and usefulness in his works on the Technology Acceptance Model (TAM) [139]. He stated that ease of use is a prerequisite to ensure usefulness [139]. Hence, the benefits of using the DIFFC model should outweigh the required effort to apply the phase of the model in the feedback cycles. In research by Hardgrave et al, (2003) on the factors that determine the intention of software developers to adopt a methodology, the authors confirm that the perceived usefulness of a software development approach positively correlates to the intention to use that particular approach [140]. According to Green et al. (2005), quality and productivity are "the most frequently claimed benefits of SPIs that make them useful to organizations" [101, p. 544]. In the context of the research by Green et al. (2015) *productivity* refers to developer productivity, which includes aspects such as the reduction of defects preceding a release, and time spent on development, among others [101]. Whereas, *quality* refers to the software quality using metrics such as robustness, maintainability, and code quality [101]. Since we are applying these variables to our validation research that focuses on documentation, we define the variables using the following statements in Table 7.1. With these variables, we design the guidelines for observation notes and validation survey. The statements accompanying the variables are then used as a basis for the observation and survey.

Table 7.1: An overview of variables and statements for the experiment and validation survey.

Variable	**Statement**	**Original Source**
Ease of use	I found it easy to apply the phases of the DIFFC model to the Scrum events of my team.	[139]
	The guidelines provided for each phase of the DIFFC model were clear and understandable.	[139]
	I found the DIFFC model to be flexible to implement.	[139]
	It was easy for me to become skilful at using the DIFFC model	[139]
Productivity	The use of the DIFFC model has speeded up the process of updating documentation within the past sprint.	[141]
	The use of the DIFFC model has made documentation more measurable within the project.	Researcher
	The use of the DIFFC model has increased my productivity with regard to keeping documentation up-to-date.	[141]
Quality	The use of the DIFFC model has enhanced the quality of documentation within the project.	[141]
	The use of the DIFFC model has decreased the amount of forgotten documentation within the project.	Researcher
	The use of the DIFFC model has clarified exactly what needs to be documented per user story.	Researcher
	The use of the DIFFC model has improved the overall quality of the Sprint Planning, Daily Scrum, and Retrospective.	Researcher
	The use of the DIFFC model has made me more conscious of documentation quality.	[101]
Usefulness	Using the DIFFC model improved the state of documentation within the project.	Researcher
	Using the DIFFC model added more structure to the way we approach documentation as a team.	Researcher
	The DIFFC model integrated well into our way of working as a team.	Researcher
	I found the activities prescribed by the DIFFC model useful in my job.	[139]
Use	I would keep using the activities of the DIFFC model in my team.	Researcher
	I would recommend the DIFFC model to other teams within the company.	Researcher

7.2 Experimental Setup

The experiment consists of three parts, preparation, execution, and evaluation. We present the preparation of this experiment. Firstly, the success of this experiment largely depends on the willingness and involvement of the project teams, and the timing of the feedback cycles of these teams. For instance, whether the Retrospective is held per sprint or after every two sprints affects the timing of this experiment. Moreover, the availability of the key roles is an essential success factor of this experiment. For example, implementing the DIFFC method without the Scrum Master present may lead to the risk of incorrect implementation due to the possible lack of guidance during the agile ceremonies of the team. In order to mitigate these risks, we require the participating teams to satisfy the following prerequisites:

- The willingness to participate and apply the (applicable) changes proposed in

the DIFFC method to the existing processes.

- The availability of key roles such as Scrum Master or the Team Manager.
- An adequate representation of the team. There should be enough team members throughout the timeline of the experiment, such that the various roles within the team are represented.

The initial plan was to apply the treatment to the four agile teams in P1 and P2. However, due to resource constraints such as the availability of team members and timing reasons both teams in P1 could not participate in the validation study. P1 had transitioned to the maintenance team within C1, the project has been down-scaled to a few people, and the focus was solely on preparing for their go-live and their goals and timeline were not suitable for this SPI. For P2, the key team members who needed to be present for this observation were on vacation until mid-August. At that time, we found out that the team has shifted from a bi-weekly to a tri-weekly sprint. Even though P2 had met all the prerequisites and was willing to experiment with the model, the timeline of their tri-weekly sprint beginning on 1st September and ending on 22nd September conflicted with the deadline of this thesis. Therefore we conducted a preliminary validation on this case by studying the application of the DIFFC model to the Retrospective (sprint n), and Sprint Planning (sprint n + 1). Due to resource constraints within the project, the Scrum Master of P2 was willing to experiment with P2T1 and then based on its effectiveness, he will apply it to P2T2 in the future. Therefore, they agreed to participate in this validation study with P2T1.

To conduct an in-depth validation study, we selected another multi-team project within C1 which is referred to as P3 in this research. P3 satisfied all the aforementioned prerequisites and has a similar structure to P1. The project is made up of three teams divided based on functionality. Each team consists of 6-8 members. The entire project develops a financial solution for applying for loans and budgeting for the public sector. The team is geographically distributed, with members in Europe and Asia. Hence, the need for communication and documentation is as essential as in the case of P1. P3 also adopts the C1M approach to the project, giving it a similar structure and organisation as P1. The documentation artefacts used by P1 are also very much similar to that of P3, in essence, we can say the issues facing documentation by P1 described in Section 5 are comparable. The main difference in the timeline of this experiment is that the project has sprints of 4 weeks rather than the biweekly sprints observed in P1. Also, due to the conflicting schedules of the meetings of two of the teams, we are only able to conduct the experiment on two of the teams within P3, which are referred to as P3T1 and P3T2.

Similar to the case studies conducted in the Problem Investigation phase, we inform participants of their rights and present a consent form as specified in Appendix E, to the participants who have not signed this form yet. In Table 7.2, we present an overview of the tasks for the experimental setup as well as the subjects involved. Prior to this, we prepare a presentation explaining the proposed DIFFC model as well as the proposal for validation research within each team.

In our PDD notation and the practitioners' of the DIFFC method, we proposed the gathering of data and generating insights in the reflecting phase. However,

Table 7.2: An overview of preparations tasks for the treatment validation research

Task Overview	**Estimated Duration**	**Subject(s)**
Schedule a meeting with the Scrum Masters to explain the proposal.	30 minutes	Scrum Masters
Organise a meeting to discuss the proposed framework and the outline of the experiment.	30 minutes	Scrum Masters
Select a Retrospective game.	-	
Agree upon the frequency of the Planning, Monitoring, and Reflecting phases for the experiment.	-	Scrum Masters
Ask the team to plan for the first reflection by gathering software project data on documentation debt within the team. The aim of this task is for the Scrum master to identify what to show to the team to aid the process improvement discussions.	1 hour	Scrum Masters
Organise a meeting to introduce the proposed framework and the selected retrospective game to the team.	30 minutes	Team Members
Send detailed guidelines for each phase of the experiment to the Scrum Masters. (See Appendix F)		Scrum Masters

even though they are in one phase, we split these actions into preparatory and executable actions for the treatment validation due to the timing of the experiment and the expected learning curve. In order to avoid the team waiting for the data to be generated and visualised, it is best for the Scrum Master to prepare for the Retrospective by having the data visualisation ready before the meeting to save time.

7.3 Experimental Execution

During the execution of the experiment, the researcher will have an observatory role, whereby the team applies the treatment model based on the instructions discussed in this section. The researcher records the behaviours and perceptions of the team members towards the application of the DIFFC model based on the variables outlined in Table 7.1. The observations will be written per meeting and analysed qualitatively. The approach of thematic analysis, as used in the Problem Investigation phase, is reused for the analysis of the recorded notes. The notes are analysed to search for patterns of the variables: *ease of use*, *productivity*, *quality*, *usefulness*, and *use*. Subsequently, using the insights from SPD, we compare the pre (sprint n) and post (sprint n+1) results of the variables outlined in Section 7.1. These variables will give insights into whether the DIFFC model has contributed to reducing documentation debt and improved documentation processes within the teams.

Our proposal for the experiment with P3T1 and P3T2 (longitudinal case) is as follows. The Planning, Monitoring, and Reflecting phases of the DIFFC model will be applied in the Sprint Planning, Daily Scrum, and Retrospective, respectively. We investigate the application of this framework in two sprints as

shown in Figure 7.2. The ideal situation would have been to observe the team for three sprints, but due to the fact that the sprints have a duration of four weeks and the timeline of this research, the researcher decided to observe two sprints. The observation will be conducted from the current iteration (Sprint n). Here, the experiment begins by applying the activities of the Reflecting phase described in the DIFFC model for practitioners, as shown in Table 7.3. The main activities are using data to generate insights and stimulating the team to generate process improvement actions. The reason why we start with the Reflecting phase is to give the team the opportunity to reflect on the state of documentation within their project and create awareness about the expectations of the various stakeholders within the team towards documentation. Also, it allows us to compare the data at the starting iteration with the data in the last iteration of this experiment (Sprint n + 1). Moreover, the set-up for the experiment of P2T1 (preliminary case) is similar to the aforementioned of P3. The only difference is we experiment with the application of the Reflecting phase to the one Retrospective (Sprint n) and one Sprint Planning (Sprint n + 1).

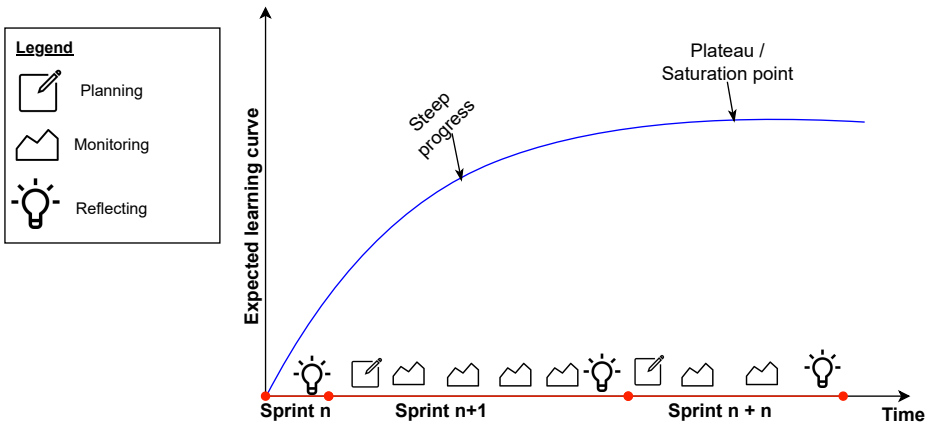


Figure 7.2: An overview of the timeline of the application of the various phases of the DIFFC model within the experiment of the longitudinal study.

Table 7.3: An overview of experimental tasks for the Retrospective.

Activity	**Task Overview**
Set the stage	Proceed with any informal discussions the team already have at Retrospectives.
	Discuss individual perceptions of the previous sprint.
	Review improvement actions from the previous retrospective.
Generate insights	Visualise and present the data to the team.
Collect improvement actions	Decide on what problem to address from the data. For example the amount of documentation debt.
	Play a retrospective game. For example the role-expectation matrix.
	Identify possible improvement actions
Assign improvement actions	Assign process improvement actions to team members.

Using the game-based approach, we select a game suitable for the nature of the team and experiment. The *role expectation matrix* retrospective game as

introduced in Section 6.5.3, is selected for this step, as it is recommended by Mesquida et al. (2019) for use dueing the norming stage of group dynamics, and it helps set the expectations of the team members towards each role to address the issue of documentation debt. Instructions for this Retrospective game can be found via the FunPMBOX with this link[1]. Also, we contacted the authors of the research by Mesquida et al. (2019), which we used as a basis for the game-based approach to Retrospectives. The authors provided us with more material on the Retrospective games which are used to aid the preparation of the retrospective games. We designed an interactive board for the Retrospective to facilitate a hybrid approach to the meeting. The interactive board can be found in Appendix G.

Furthermore, the awareness of the DIFFC model and the process improvement actions elicited from the team will set the stage for the Planning, and Monitoring activities for the next iteration (Sprint n + 1). In Sprint n + 1, we recommend the occurrence of the Planning phase at the start of the iteration. The planning activities are done per user story to ensure that documentation is made measurable and operationalised. The proposed activities of the Planning phase on a user story level are to be incorporated in the Sprint Planning meeting for Sprint n + 1. Furthermore, we recommend at least four occasions of process monitoring for the four-week sprint, during the Daily Scrum. Whereby the progress of the documentation tasks within the user stories, and the process improvement actions proposed during the Retrospective of Sprint n, are reported as well as any issues or dependencies preventing the successful completion of these tasks. Finally, we conclude with an occurrence of the Reflecting activities, whereby the perceptions of the team members towards the change are discussed, and then a data-driven approach is taken for the Retrospective as done in Sprint n. Here, we select another Retrospective game to generate new ideas and encourage active participation. We expect to observe feedback about the changes introduced in the previous iteration due to the introduction of the DIFFC model. From the Retrospective of Sprint n, we saw that the DIFFC model was effective in creating awareness within the team about the state of documentation. The next step was to stimulate the team to derive more process improvement actions to address the documentation issues. For the Retrospective of Sprint n + 1, we select the *who-what-when* retrospective game. As it is recommended as an approach to define and plan the tasks to be carried out by the team in the upcoming period [132].

For the experiment with both teams in P3, we expect to see a steep learning curve of the team members in the beginning to a point of saturation, where the team is familiar with the activities of the DIFFC and the process of documentation is improved within the team. After this experiment, for instance in Sprint n + n, it is up to the team, as a self-organising team to select which phases to use for each iteration, which retrospective game to use, and when and how often to use it. We present timelines of the frequency of observation in Tables 7.4 and 7.5 for the selected teams in P3 and P2, respectively.

[1]https://funpmbox.github.io/jekyll/update/test/2014/04/20/que-esperas-de-mi.eng.html

Table 7.4: An overview of feedback cycles to observe the longitudinal application of the DIFFC model in P3.

Phase (Feedback Cycle)	**Team**	**Sprint**	**Date scheduled**	**Duration**
Reflecting (Retrospective)	P3T1	Sprint 26	26th July 2022 at 11:00.	1 hour
	P3T2	Sprint 26	26th July 2022 at 12:30.	1 hour
Planning (Sprint Planning)	P3T1	Sprint 27	27th July 2022 at 10:00.	1 hour
	P3T2	Sprint 27	27th July 2022 at 11:00.	1 hour
Monitoring (Daily Scrum)	P3T1	Sprint 27	28th July 2022 at 09:15.	15 minutes
	P3T2	Sprint 27	28th July 2022 at 09:30.	15 minutes
	P3T1	Sprint 27	5th August 2022 at 09:15.	15 minutes
	P3T2	Sprint 27	5th August 2022 at 09:30.	15 minutes
	P3T1	Sprint 27	12th August 2022 at 09:15.	15 minutes
	P3T2	Sprint 27	12th August 2022 at 09:30.	15 minutes
	P3T1	Sprint 27	19th August 2022 at 09:15.	15 minutes
	P3T2	Sprint 27	19th August 2022 at 09:30.	15 minutes
Reflecting (Retrospective)	P3T1	Sprint 27	23rd August at 11:00, 2022	1 hour
	P3T2	Sprint 27	24th August at 14:30, 2022	1 hour

Table 7.5: An overview of feedback cycles to observe the preliminary application of the DIFFC model in P2T1.

Phase (Feedback Cycle)	**Team**	**Sprint**	**Date scheduled**	**Duration**
Reflecting (Retrospective)	P2T1	Sprint PI22.3-5	1st September 2022 at 11:00.	1 hour
Planning (Sprint Planning)	P2T1	Sprint PI22.3-5	1st September 2022 at 14:15	1 hour

7.3.1 Validation Survey

The survey was given after the longitudinal study to the participants from P3. The survey entails Likert scale-type questions. The questions are based on the variables and the statements in Table 7.1. To ensure consistency and the ease-of-use of the survey, we adopt a 5-point Likert scale for all the Likert scale types of questions. For example, the scale ranges from Strongly Agree to Strongly Disagree. The questions are for each phase of the DIFFC model. We present the survey in Appendix H. The survey was given to the participants at the end of the retrospective for sprint n +1, therefore the survey was only given in the longitudinal study. Out of 17 participants (8 from P3T1 and 9 from P3T2), 13 participants (8 from P3T1 and 5 from P3T2) filled in the survey.

7.4 Findings (Longitudinal study)

We discuss the results of the experiment and survey to validate the DIFFC model as a means to address documentation debt and improve documentation processes within P3.

7.4.1 Retrospective Sprint n (26)

The first studied Retrospective meetings were divided into two parts, and both teams took slightly different approaches to the Retrospective. For the first part of the Retrospective P3T1, the Retrospective board in Azure DevOps was used with three columns to discuss the following about the last sprint.

- What did we do well?
- What could be better?
- What needs our attention?

This part of the meeting was time-boxed, allowing time for the documentation process improvement part of the meeting. We saw that there were some issues in the third column concerning the outdated documentation as a result of the change requests. For example, the original scope is not clear after a change is implemented. Azure DevOps provides the Collect, Group, Vote, and Act steps for Retrospectives. *Collect* is where the team post their ideas in the form of notes on the virtual board. *Group* is where similar topics are grouped to make it easier for voting. Then, the team had 3 minutes to vote, where each team member was given 3 votes to vote on the most important notes. Afterwards, the last step, *Act* is used to highlight the most voted points and have a discussion and possibly make improvement actions, also referred to as retrospective actions.

The second part of the Retrospective involved using the guidelines of the Reflecting phase of the DIFFC model. The researcher prepared a Miro[2] interactive board beforehand with the roles in each team to facilitate the Role Expectation Matrix game. A similar approach to the Retrospective was taken by P3T2. However, the two parts of the meeting were organised the other way around. They began with the documentation part of the Retrospective using the DIFFC model and then discussed the perceptions of the team using the Retrospective board in Azure DevOps. The team also followed the Collect, Group, Vote, and Act steps provided by Azure DevOps.The Retrospective Board entails four columns which are explained as follows.

- Keep (What should we keep doing?)
- Add (What should we incorporate into our way of working?)
- Less (What should we do less of?)
- More (What should we do more of?)

From this part of the observation, we see evidence of the discussion of perceptions about the previous sprint which relates to the first activity of the Reflecting phase.

[2]https://miro.com/

7.4.2 Sprint Planning Sprint n + 1 (27)

The Sprint planning meetings for P3T1 and P3T2 were scheduled for 2 and 3 hours, respectively. The researcher observed the first hour of each Sprint planning, due to the conflicting schedules of both teams and the timeline of this research but, was informed about the decisions made and the use of the DIFFC model throughout the rest of the meetings. For both teams, a proposal of the sprint backlog was set up and discussed with the team during the meeting. The guidelines were given to the Scrum Masters of the team to incorporate the Planning phase of the DIFFC model.

During the Sprint Planning of P3T1, tasks from the user stories that are not finished are moved from the previous sprint to the new sprint. Then, there was a discussion about the tasks and what needs to be done. Links to documentation are checked for some of the user stories. And the team members give estimates in hours per task. It was difficult to see the guidelines being applied in the first hour observed, as there were a lot of tasks as part of a huge user story to be estimated. However, the Scrum Master reported that the guidelines were used in the later part of the Sprint Planning for a change request. Here, they added two tasks, one to update the documentation and the other to review the documentation. The scope of the documentation was also discussed among the team and written down in the task.

In the case of P3T2, the setup of the Sprint Planning was very similar to that of P3T1. The team went through the tasks to discuss what it entails and what work needs to be done, afterwards, estimation is given in hours. We saw the guidelines of the Planning phase of the DIFFC model, being applied in the first hour of observation. Tasks were added to update the migration documentation for the migration PBI. But the scope was not defined, and a rough estimate was given.

7.4.3 Daily Scrum meetings Sprint n+1 (27)

We observed a 15-minute Daily Scrum in the first week of Sprint 27 for P3T1 and P3T2 separately. The order of the meeting was based on the alphabetical order of the names of the team members using the Sprint Board in Azure DevOps. The team members discuss the following questions per turn.

- What did I do yesterday?
- What will I do today?
- Are there any impediments?

They report on the tasks they are working on or have worked on. Hence, if the task is not a documentation task, then, they do not report on documentation and this was the case of the first observed Daily Scrum meetings. Additionally, process improvement actions were not discussed in these meetings as there was no progress to report on those as yet. The situation was similar to the observation of the subsequent Daily Scrum meetings. This shows that *the effectiveness of the guidelines of the Monitoring phase depends on the productivity in the Planning phase.* If documentation tasks are defined in the Planning Phase, the progress of these tasks is reported in the Monitoring phase.

7.4.4 Retrospective Sprint n + 1 (27)

The Retrospectives for both teams in P3 took a similar approach to that of the previous sprint. As mentioned in Section 7.2, we selected the who-what-when retrospective game to stimulate the idea generation of process improvement actions. Here, the documentation part of the retrospective was scheduled for the second part of the meeting. For P3T1, due to the large number of backlog items to address from the previous sprint, they also applied the who-what-when game to address those topics. There were a total of seven topics of which one addressed the issue of incomplete documentation tasks in the user stories. From the guidelines for the sprint planning, documentation tasks were added that were not completed, and those hindered the completion of the user stories towards the end of Sprint 27. As their chosen approach to include other topics was not in line with the guidelines, it reflected in the results whereby the focus was not on documentation and there was only one process improvement action for the documentation. Therefore, those results were not used for further analysis. Yet this confirms that *the success of the application of the DIFFC model largely depends on the willingness of the team to apply the guidelines and also prepare SPD concerning documentation.* If the SPD concerning the documentation was gathered and visualised, the focus on documentation would have increased in this exercise.

On the other hand, the approach of P3T2 was a success thanks to the willingness of the team to apply the guidelines and to schedule enough time for it. 30 minutes were scheduled for this part of the meeting. The goal of the exercise was to ask the team members to write down what was lacking in the documentation. The discussions about the topic were interactive and the team had a high involvement in idea generation. Seven aspects of the documentation were identified as areas for improvement, including the Detailed Design artefact for the loans functionality, the referencing to the old design in user stories (when the old design is no longer correct), features without descriptions and links to specific documentation, and the suggestion of a separate teams channel to announce documentation updates.

7.5 Findings (Preliminary study)

The Retrospective for Sprint PI22.3-5 was organised as a pilot to apply the DIFFC model. Prior to the Retrospective, the Scrum Master had prepared the interactive environment, a Mural[3] board for the role expectation matrix. As part of the preparation guidelines, we asked the Scrum Master to prepare the SPD with regard to the issues facing documentation and decide upon what needs to be addressed. The task was assigned to the Lead Developer, however, this was forgotten. To continue, the team brainstormed for ideas on what is going wrong in the documentation and decided to address the question of 'How complete are the user stories?' Therefore, the exercise was to address what each role expect from the others to ensure the completeness of user stories. We see a similar pattern with the teams in P3, whereby the SPD was not prepared before the meeting because it was forgotten or due to time constraints.

[3]https://www.mural.co/

Subsequently, we observed the Sprint Planning after the Retrospective for P2T1. The Sprint Planning took a different approach than that observed in the teams in P3. Here, the user stories had already been estimated and at this meeting, they were being assigned to the developers and ensuring that the planning, timing, and dependencies of the user stories are addressed. Therefore, the Planning Phase of the DIFFC model will be more suitable to apply at the Refinement where the estimations are made. However, tasks per user story are created by the assigned developer after the Sprint Planning, so the tasks are not created and discussed upfront.

7.6 Results: Measure of effectiveness of the DIFFC model

We address the variables based on the experiment and the results from the validation survey introduced in Section 7.3.1.

7.6.1 Ease of use

During each Retrospective session, the ease of use of the model was observed by how easy it is for the team to adapt to the Reflecting phase of the model. At the start of the documentation part of the Retrospective (in the case of P3), or the Retrospective (P2T1), we see a *similar pattern of unfamiliarity with the guidelines where questions are asked upfront about how the interactive environment works, or how the Role Expectation Matrix is played.* Even though there were guidelines about the model and the Role Expectation Matrix. But after the explanation was given, the team gained familiarity with using the tool and interacting with the Role Expectation matrix.

Two of the guidelines of the Reflecting phase ensure the gathering of software project data and generating insights to be used as a topic for discussion among team members. What we saw is that *the teams did not have the time to prepare and gather data for the Retrospectives.* The Scrum Master of each team in P3 decided that the goal was to address issues facing documentation within the team, for the Retrospective of Sprint n. Not only was time an issue in preparing the data, but it turned out that there is not always SPD about documentation debt, because there might be no tasks that were forgotten or moved back to the backlog. But the documentation artefact lacks updates, and it needs to be analysed before having data about how much work is needed to update it, for instance. A similar issue was also identified in P2, whereby the user stories do not have documentation tasks. For P3T1, there were two matrices created, which addressed the topics of 'Underestimating the number of hours for tasks', and 'Updating or adding documentation on a PBI (Product Backlog Item)'. Whereas P3T2 addressed the topic of 'Improving documentation'. During the second Retrospective session (Sprint 27), P3T1 used the what-who-when game to address the issues from the first part of the Retrospective. This reduced the focus on documentation as discussed in Section 7.4.4. P3T2 on the other hand address the subject of lacking documentation as proposed by the Scrum Master.

During the Sprint Planning for P3T1, the Planning activities from the DIFFC model were not used as much as we expected due to the large among of tasks

in the sprint and the time needed to estimate each task. But for both teams in P3, *we saw that the guidelines that were sent to the Scrum Master were clear*. Indeed, the researcher did not need to explain how to use the steps. In P3T2, the Scrum Master started a discussion with the team members about their perceptions of the current state of documentation within the team. The issue for P3T2 is that *the documentation is out of sync with the change requests*, and this is the same situation for P3T1 as well. Both teams added tasks for the change requests, where applicable for updating and reviewing the documentation. An example is the addition of a task to update the documentation with the specific name of the documentation as well as the scope of what should be documented. Here, the business consultant also elaborated upon the documentation task and the team also added a review task for the documentation. For P3T2, besides the addition of documentation update and review tasks, there were also discussions about the documentation debt and how it can be addressed in the upcoming sprint, therefore some other documentation tasks relating to documentation debt were added to the Sprint Backlog.

Survey results

From the 13 responses, we discuss the results of the ease of use of the Planning (PP), Monitoring (MP), and Reflecting (RP) phases in Figure 7.3.

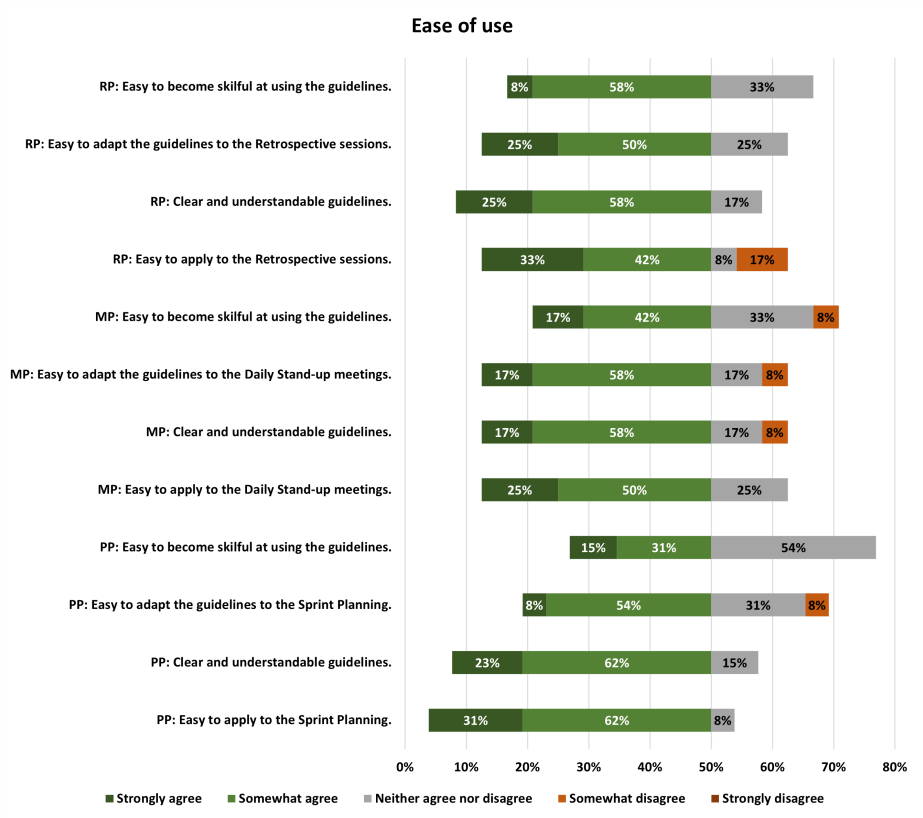


Figure 7.3: Survey results of the *ease of use* of the phases of the DIFFC model.

From the survey, we see that for most aspects of ease of use, the results were positive. Also, more than half of the respondents agreed that it is easy to adapt the guidelines to suit the Sprint Planning. However, the perception of becoming skilful at using the guidelines of the Planning phase is rather low with the agreement of only 6 respondents.

7.6.2 Productivity

For P3, *we see a difference across both teams in how interactive the Role-Expectation Matrix was. A factor in this is the allocated time.* P3T2 planned this as the first activity of the Retrospective, meaning they had about 30 minutes allocated for it, whereas P3T1 performed this as the last activity, with about 30 minutes allocated, but to address both topics. In Figures 7.4 and 7.5, we see the Role expectation matrix used for P3T2 and that used for the documentation topic of P3T1, respectively. With regard to the assessment of productivity, it was not clear from the Retrospective of Sprint n whether the use of the model has helped speed up the process of writing documentation. This will be assessed in the Retrospective of Sprint n+1, as well as the survey. Moreover, assessing whether documentation has been made more measurable within the project can be possible by identifying process improvement actions towards addressing the issues found in the documentation. Team P3T1 added an improvement task to adapt the template of a user story in Azure DevOps, and add a documentation specification section. In P3T2, there were no improvement actions derived. *The essence of this approach was to facilitate the discussions, identify issues facing documentation, and create awareness of the expectations of team members towards one another.*

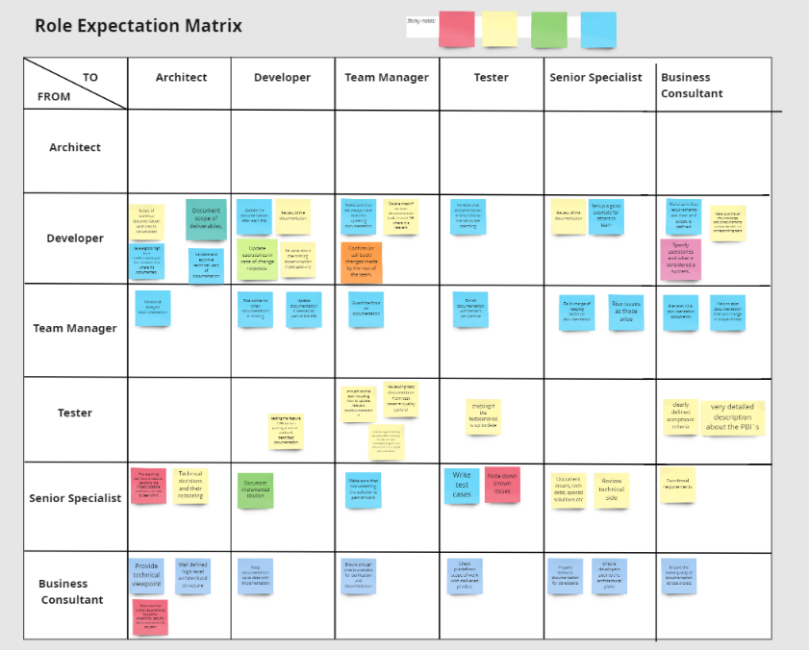


Figure 7.4: An overview of the result of the role-expectation matrix by P3T2.

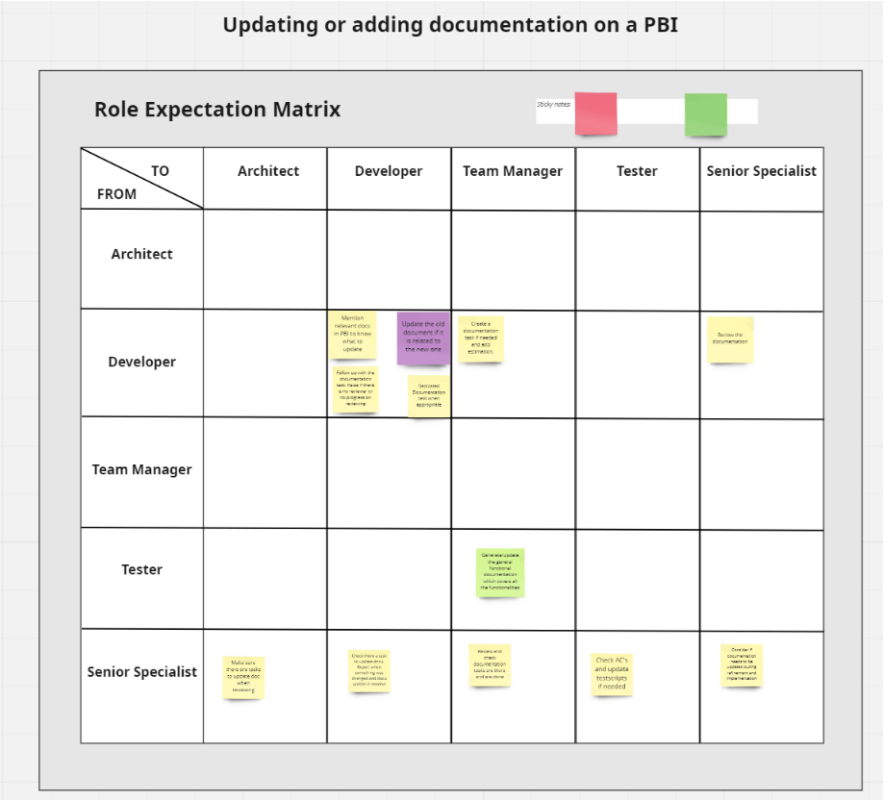


Figure 7.5: An overview of the result of the role-expectation matrix for updating documentation by P3T1.

Similarly, the Retrospective of P2T1 facilitated discussions and created awareness to address the issue of incomplete user stories. The level of interactivity was very high, it is interesting to see that *since 1 hour was allocated for the experiment, the team had more time to discuss the ideas* and even made two process improvement actions to be incorporated in the upcoming Sprint. The actions are as follows.

- To ensure that the value of Product Risk is clarified by the tester for each user story. This corresponds to the expectations from the Developers and Lead Developer to the tester.
- To ensure that the Developers take into account the updates made to user stories but also update use stories, using the comments section where the implementation deviates from the requirement.

During the Sprint Planning, *both teams in P3 added tasks to update and review documentation as well as defined the scope of what should be documented.* This had made documentation more measurable in the project by ensuring both teams add tasks for documentation with estimates for not only updating the documentation but also reviewing the documentation.

Survey results

We present the survey results for the perceptions of productivity from the longitudinal study. The results are presented in Figure 7.6.

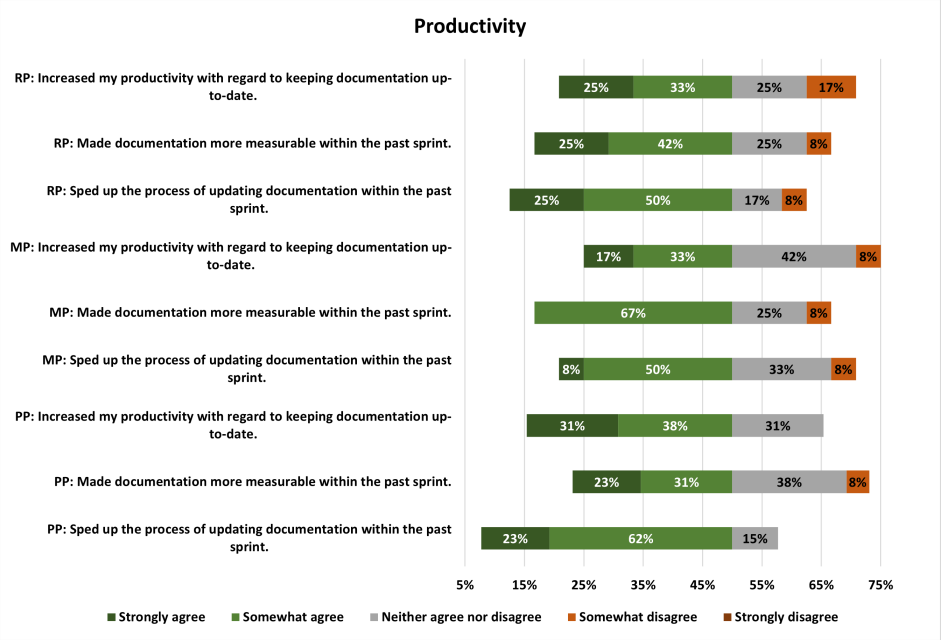


Figure 7.6: Survey results of the *productivity* of the phases of the DIFFC model.

From the results, we see a high agreement on how the guidelines of the Planning phase have helped speed up the process of updating documentation within the team. Similarly, we see 9 respondents agree with this same aspect for the Reflecting phase. About half of the respondents did not agree or were neutral about whether the Monitoring phase has increased productivity with regard to keeping the documentation up-to-date. A similar result is also seen in whether this same phase has sped up the process of writing documentation.

7.6.3 Quality

Considering the increased awareness of the quality and issues facing documentation, the Reflecting phase of the DIFFC model is likely to contribute to improving the quality aspects of the documentation. Also, the addition of a process improvement action by P3T1 to improve the template of the user story is likely to ensure that the team clarifies exactly what needs to be documented. P2T1 also added process improvement actions which are likely to improve the quality of user stories and ensure traceability of functionality after the user stories have been developed.

During the Sprint Planning, the discussions about the outdated documentation within P3 as a result of the change requests were certainly an eyeopener for the team members. As discussed in Section 7.6.2, the addition of documentation update and review tasks also helped clarify what exactly needs to

be documented and this approach is likely to reduce the amount of forgotten documentation within the team. Also, *the awareness made the team more conscious of documentation quality and aware of the documentation that is missing updates.*

Survey results

We present the survey results for the perceptions of quality from the longitudinal study. The results are presented in Figure 7.7.

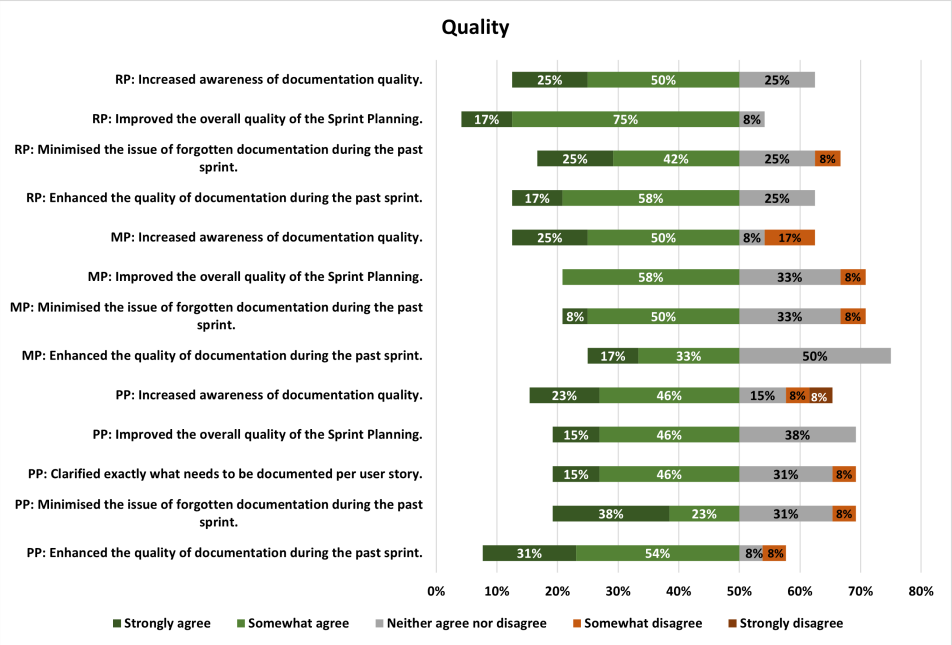


Figure 7.7: Survey results of the *quality* of the phases of the DIFFC model.

We see a range of approximately 80-50% of respondents agreeing with the various aspects defined for the three phases of the model. The lowest agreement with the given statements is the aspect of whether the Monitoring phase has enhanced the quality of the documentation, followed by whether this phase has minimised the issue of forgotten documentation during the past sprint.

7.6.4 Usefulness

At the end of the first Retrospectives for P3, there was a short feedback round where the team members could give feedback on the use of the reflecting phase of the DIFFC model during the Retrospective. *In both P3T1 and P3T2, there were positive remarks about the use of the model and the role expectation matrix that created awareness among the team members about the state of the documentation.* Also, it generated a lot of ideas about how the team can improve upon documentation. For P2T1, there was no feedback round due to the timing, but there were remarks about how useful the guidelines were in creating awareness of the issues facing the user stories.

As discussed in Section 7.6.1, *the teams had some initial difficulty using the prescribed tool.* Also, *it did not integrate well into their way of working as a team for the teams in P3.* Using the free version of the Miro board, it was not possible to export the result of the role expectation matrix in a readable format. Therefore, the Scrum Masters had to manually summarise the noted points and email the team about it. Using their existing way of conducting Retrospectives, Azure DevOps provide an easy and clear export functionality, and it is easy to make process improvement actions in the 'Act' stage of the Retrospective. However, with this tool, it is not possible to directly create process improvement actions as tasks in Azure DevOps. Therefore, for the Retrospective sessions for Sprint n + 1 for P3, the researcher asked the Scrum Master to prepare the board in Azure DevOps, as this is an environment they already used for the Retrospective prior to the experiment. Hence, this addressed the issue of integrating well into the way of working. However, not all the recommended Retrospective games can be prepared using the column-based interactive board in Azure DevOps.

With the guidelines in place for the teams to review user stories and add tasks to update and review documentation, we believe this initiative will be useful to the team by adding more structure to the way the team approaches documentation.

Survey results

We present the survey results for the perceptions of usefulness from the longitudinal study. The results are presented in Figure 7.8.

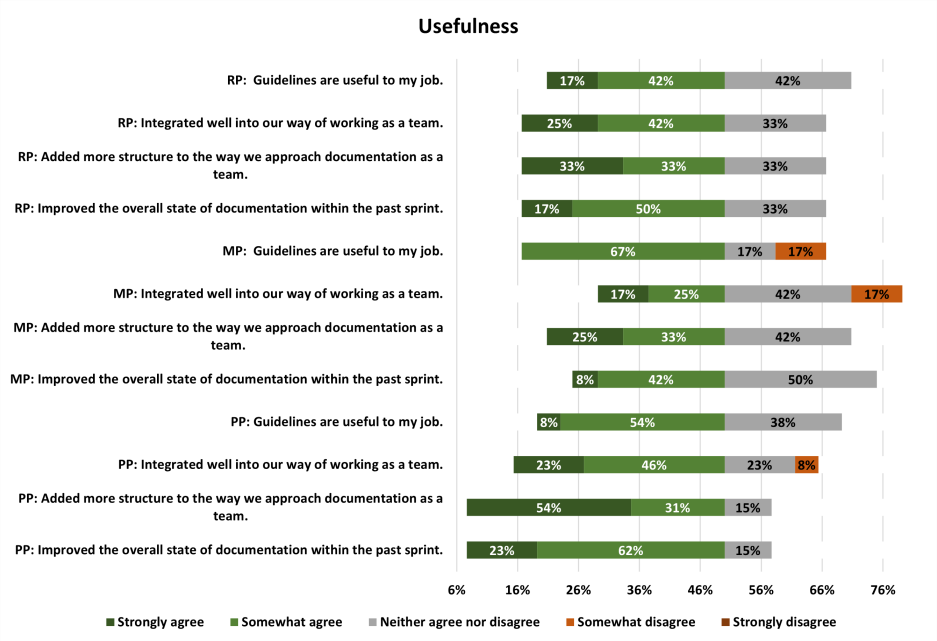


Figure 7.8: Survey results of the *usefulness* of the phases of the DIFFC model.

From the results, 11 respondents agreed that the guidelines of the Planning

phase have improved the overall state of documentation within the sprint. The same number of respondents also agreed that the use of the Planning phase added more structure to the way documentation is approached by the team. Considering the aspect with the lowest rate of agreement, we see that only 42% agreed that the Monitoring phase integrated well into the way of working as a team. This may relate to our findings from observing that *the guidelines of the Monitoring phase are only useful when there is progress on the documentation or process improvement tasks prior to the Daily Scrum.*

7.6.5 Use

From the observations, *it is not enough data to verify whether the team will keep using the activities of the DIFFC model.* Even though there were positive remarks as mentioned in Section 7.6.4, it is not enough to make a solid conclusion. There were also some improvement points for the Retrospectives for Sprint n + 1 for P3. That is to select a Retrospective game with a lesser duration than the role expectation matrix, therefore we selected the who-what-when game. Also, the majority of the team has a Developer role, so discussing the writing notes per row took longer and it was sometimes difficult to know who to ask to explain the tasks because you cannot see who wrote what if they do not volunteer to explain their written note. The survey, therefore, gives a much better perspective of whether the various phases of the model will be used or recommended to other teams within the company.

Survey results

We present the survey results for the perceptions of use from the longitudinal study. The results are presented in Figure 7.9.

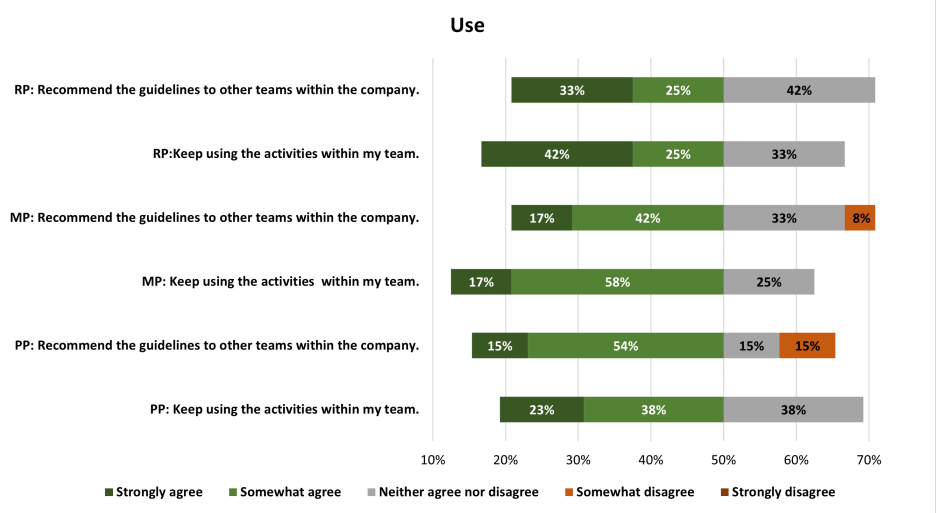


Figure 7.9: Survey results of the *use* of the phases of the DIFFC model.

From the results, the lowest rate of agreement (only 7 participants) concerns the Monitoring phase, and whether the guidelines of this phase will be recom-

mended. This corresponds to the lower rates of agreement we saw for the aspects of usefulness for this phase. In general, *more than 60% of the respondents will keep using the activities of the various phases of the DIFFC model.* Even though the Retrospectives were successful in creating awareness of the issues found in the documentation, only 7 respondents agreed that they will recommend the guidelines of the Reflecting phase to other teams. There was no feedback on this, but the remarks on the Reflecting phase from the survey are presented in Table 7.6.

Table 7.6: Remarks for the phases of the DIFFC model from the validation survey.

Phase	**Feedback**
Reflecting	Applying this phase helped to put more focus on the missing documentation.
Reflecting	I believe that before [the application of the Reflecting phase] the topic of missing or incomplete documentation was rarely brought up.
Planning	Having documentation as part of implementation is not something that was not done before. However, it was often forgotten. Having this as an explicit step during planning brings valuable attention to documentation and lets the team look critically at what is missing.
Planning	Due to the documentation not being up to date, it is hard to measure effectiveness. Definitely, I could see improvements in team focus on the documentation tasks.

7.6.6 Pre-Post Comparison

We addressed the variables with regard to insights from the SPD before the experiment (start of sprint 26), during the experiment (start of sprint 27), and at the end of the experiment (end of sprint 27). This data concerns only the longitudinal case as the experiment entailed the application of the DIFFC model to a complete sprint. Figures 7.10 and 7.11 present an overview of the data received from the Scrum Masters of P3T1 and P3T2, respectively.

During the experiment, we observed that documentation tasks were added for the user stories during the Sprint Planning of Sprint 27. Compared to the previous sprint, we see an increase from 3 of these tasks to 8 tasks for P3T1. This also corresponds with the estimation from 10 hours of documentation for Sprint 26 to 34 hours. Only 3 out of the 8 documentation tasks were completed. The completed effort for those 3 documentation tasks was 12 hours in total. This also reflects the discussion in the Retrospective of Sprint 27, that the user stories were not closed due to the incomplete documentation tasks. It is interesting to look into why most of the documentation tasks were not completed, and how can the DIFFC model be enhanced to address this issue in future research.

In the case of P3T2, we see some similarities such that the number of planned documentation tasks increased from 2 tasks with an estimate of 10 hours to 8 tasks with an estimate of 34 hours. In contrast with P3T1, this team had completed 6 out of 8 documentation tasks, in a time span of 27 hours, with the remaining 2 documentation tasks having a total estimated effort of 7 hours. With this effort spent on documentation during the sprint, we also observed

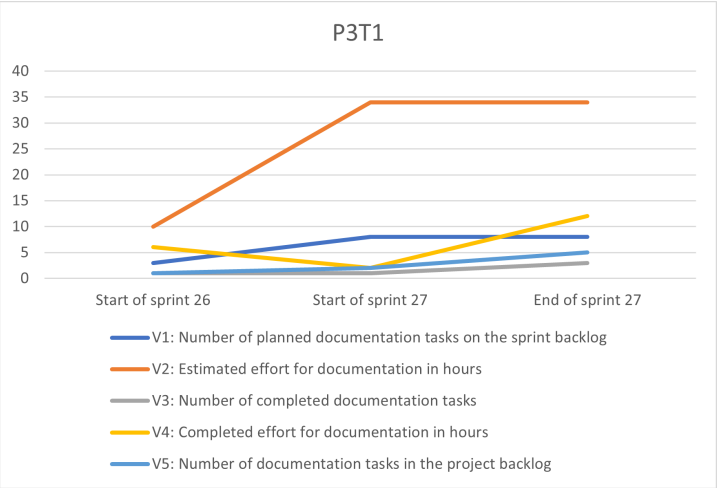


Figure 7.10: Results of the variables for pre-post comparison in P3T1.

that during the Retrospective of Sprint 27, there were more specific insights from the team members with regard to what can be addressed to improve the current state of the documentation.

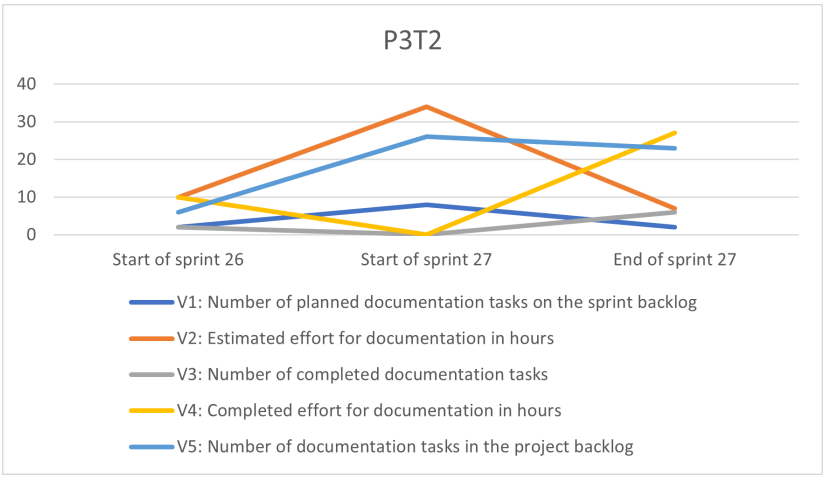


Figure 7.11: Results of the variables for pre-post comparison in P3T2.

For the last variable, V5, the numbers of P3T1 are not significant enough to draw conclusions about whether the model contributed to realising more documentation tasks to improve the current state of the documentation. For P3T2, we see that the number of documentation tasks in the product backlog increased from 6 to 26 at the start of Sprint 27. The Scrum Master explained that this occurred due to a new change request that changes 16 flows in the system, for which a documentation task was added for each flow.

7.6.7 Evaluation of Hypotheses

The number of respondents is not sufficient enough to identify significant findings for testing the hypothesis. Therefore we evaluate the hypotheses qualitatively. For H1, we see that the quality aspects associated with the various phases of the DIFFC model, such as creating awareness of the current state of the documentation, generation of ideas and creating process improvement actions to address these issues are benefits that may contribute to high perceptions of usefulness of this model. This is similar to that of H2, if the guidelines of each phase contribute to keeping the documentation up-to-date, making the documentation more measurable and increasing accountability, then it is more likely that the team members will find this model more useful. From observing the various agile ceremonies, we saw that once the guidelines are clear and understood by the team, they are more likely to embrace the study. Even though the team, members had to get familiar with the guidelines at the beginning, it became clear and the team members actively participated in most cases. This confirms the possible link with H3 from these findings.

Furthermore, the last two hypotheses discuss the effect of usefulness and ease of use on the use of the DIFFC model. Again, with the survey responses, we cannot conclude any solid response. But we see if the guidelines of the various phases are useful and improve the state of documentation, then the team is likely to adopt the model and keep using it.

7.6.8 Conclusion

To conclude we see promising results from the experiment and validation survey with regard to the effectiveness of the DIFFC model in addressing the issue of documentation debt within multi-team ASD projects. It is of course in the early stages of validation of this model, it has not been validated with a wider range of projects and not for a longer period of time (more than one sprint). We therefore cannot generalise our results to all multi-team ASD projects. But we can learn from this validation study towards improving the DIFFC model in future research.

The findings confirm that a huge part of the success of the application of the DIFFC model depends on the willingness of the team members to follow the prescribed guidelines, participate actively, and dedicate time to preparing the SPD prior to the Retrospective. However, as none of the teams prepared the SPD, it is an area of improvement for the DIFFC model to address the question of '*What happens when there is no prepared data or subject to address?*'. Also, we learnt that in practice the Planning phase may be incorporated in not only the Sprint Planning but other agile ceremonies such as the Refinement. Lastly, we observed an overlap in the phases. The effectiveness of the monitoring phase largely depends on the productivity in the Planning and Reflecting phases. Without documentation tasks or process improvement actions, the guidelines of the Monitoring phase cannot be applied effectively.

Chapter 8

Conclusions

Using the Design Science research approach, this thesis explored the impact of RE in multi-team ASD projects. In particular, we researched the aspects of scaled agile software development of SAFe and an internally created method (C1M) in practice. We identified the various activities, roles, and tools, used by the explored projects in their approach to ASD. We highlighted evidence of RE practices in scaled-agile approaches such as the use of workshops, refinement sessions, and informal communication as activities in agile requirements engineering. Here, we identified RE activities and documentation artefacts used for elicitation, analysis, specification, validation, and requirements management.

Having understood the activities involved in the scaled-agile approach and the documentation artefacts employed by the projects, we investigated the current state of documentation within the project by identifying the strengths and potential areas for improvement by considering the Information Content (What), Information Content (How), and Process Related aspects. The challenges facing documentation for the studied projects became a motivation to design a framework to fill the research gap on a prescribed approach to addressing documentation issues by improving documentation processes through feedback cycles. As a result, we introduced the DIFFC model with both scientific and industrial relevance to address this problem. Finally, we validated our proposed model by applying it to a real-world context by means studying its effectiveness in two cases. The results regarding its usefulness are promising and there are also ideas for further improvement of the model.

8.1 Answers to Research Questions

Our research question, stated as: *How can the challenges facing documentation due to Agile RE activities in multi-team software projects be addressed?* is answered as follows. The DIFFC model is proposed to address the issues facing the documentation such as time constraints, planning, establishing a clear scope, estimation, review, tracking progress, and reducing documentation debt. This framework also combines a data-driven approach with a game-based approach to improve the documentation process within the team. We elaborate upon

the aforementioned response using our seven sub-research questions which are discussed subsequently.

8.1.1 SRQ1: How is Agile Software Development (ASD) adopted within a multi-team software project?

From the explored cases, we identified two different approaches to scaled ASD which are SAFe and an internally created method referred to as C1M. We highlight the *roles*, *processes*, and *tools* used in the adoption of scaled ASD in each of the explored cases.

Roles

In Table 8.1, we outline the common and different roles across the teams. The common roles across teams are the Scrum Master and Product Owner, even though the Scrum Master role is performed by the Team Manager in P1. Also, the Product Owner in P1 is an external stakeholder, and the Analyst performs the role of the Product Owner internally, by eliciting the requirements from the client and translating them to user stories. The role of Developers, Lead Developer, and Tester exist in all teams but P2T2, while in P2T2 the role of the developer is performed by Digital Engineers. Although P2T2 does not have a Tester, testing activities are conducted by the Digital Engineers using the four-eyes principle.

Table 8.1: Overview of similarities and differences in roles across teams.

Common Roles (P1, P2T1)	**Unique Roles P1**	**All Roles P2T2**
Scrum Master	Analyst	Scrum Master
Developers		Domain Experts (Also in P2T1)
Lead Developer		Digital Engineers
Tester		Product Owner
Architect		
Product Owner		

Processes

A common feature across cases is the division of the project into two phases; the planning/elicitation phase and the DevOps phase. The elicitation phase is referred to as the clarification phase in P1 and the PI planning phase in P2. In P1, the clarification phase begins with analysing the initial project documentation, depending on the team as shown in Figure 4.2. The scope and epics or user stories are defined and planned into workshops. Prior to the workshops, the user stories are prepared together with wireframes where applicable. During the workshop, decisions are recorded and the user stories are updated. The PI planning phase of P2 is centred around the project's adoption of SAFe. During this phase, as shown in Figure 4.3, the roadmap is defined for the subsequent PIs, which define product increments or epics. These epics are then planned across the teams, dependencies across teams and features are identified, and then the features are described by the Product Owner. Afterwards, the user stories are written and defined in the Team Backlog.

The DevOps phase for P1T1 is similar to that of P1T2, whereas the DevOps phase of P2T1 differs from P2T2. We discuss the main activities in this phase. In P1, the Sprint Backlog defines the user stories proposed for the sprint. A new biweekly sprint begins with a Sprint Planning meeting. During the sprint, each team follows Scrum where the day starts with a Daily Standup and then, sprint tasks such as design, implementation, testing, and documentation are performed per user story. P2 also follows a similar adoption of Scrum on a team level. The DevOps phase begins with the Team Backlog (similar to the Product Backlog of P1T1 and P1T2). Then, the Sprint Backlog is a proposal for the sprint. A Sprint Planning meeting is also used to plan and estimate the effort for the user stories and start the biweekly sprint. Each working day, the daily Scrum is used similar to the teams in P1, however, the difference is in the activities of the sprint tasks. Before the design of a user story, analysis is performed for complex user stories. P2T1 incorporates a design task for the user stories, whereas P2T2 do not have any design tasks and moves straight to implementation after the analysis. Manual tests are performed in both P2T1 and P2T2, however in different formats, as shown in Figures 4.8 and 4.9. Afterwards, the DoD may be checked but this is done occasionally and not enforced by either of the teams. Then for P2T2, the user story is completed, whereas P2T1 has automated deployment in place in parallel with the synchronisation of the source code before a user story is completed.

All the teams use alignment meetings to communicate processes within the team, across teams within the project, or even across knowledge groups. In P1 examples of these meetings are chapter meetings, weekly design and analysis meeting, technical standup, bi- or tri-weekly architecture meetings with the client, weekly sprint update, and Refinement. The Refinement session is also adopted by the teams in P2. Other common alignment activities for both teams in P2 are the bi-weekly ART sync meeting and the weekly PO sync meeting. P2T1 also participate in Guild meetings to share knowledge across projects and teams within the company on a specific topic. Occasionally, P2T2 holds a team alignment meeting to discuss their processes as a team. Lastly, across teams, change requests are handled by adding new user stories and adjusting the sprint plan.

Tools

All teams use Azure DevOps to manage the backlog and as the main source of truth for the user stories. Additionally, Microsoft Teams is used by each team for communication. Microsoft Office tools such as Word is heavily used in P1 to write documentation as most of the documentation artefacts are prescribed by C1M and are Word processing files. In contrast, Microsoft Word is not used as much for documentation, it is mainly used for writing the Analysis Report. A SharePoint environment is prescribed by C1M for use in P1, where all the templates are documentation are stored for the project. P2T1 uses the wiki for most documentation within the team such as the explanation of the domain and the code. P1 does not use the wiki as often, they use it to aid the chapter meetings. Other tools such as DrawIO and Figma are used in P1 to model and detail requirements, whereas P2T1 uses PlantUML, to model the high-level architecture of the functionality. In Table 8.2, we an overview of the different

and similar tools used across teams.

Table 8.2: Overview of similarities and differences in tools across teams.

Common Tools	**Unique Tools P1**	**Unique Tools P2T1**
Azure DevOps	Axshare	Wiki
Microsoft Office	DrawIO	PlantUML
Microsoft Teams	Figma	
Visual Studio	SharePoint	
	Wiki	

8.1.2 SRQ2: Which requirements engineering practices can be identified in the adopted ASD approach?

The subsequent step after identifying the ASD approach of the projects was to identify evidence of *elicitation*, *analysis*, *specification*, *validation*, and *requirements management* activities.

Elicitation

The teams in the explored cases take different approaches across projects. Most of the elicitation takes place during the clarification phase for P1 and the PI planning phase P2. The technique of documentation analysis is identified in two different activities during the clarification phase of P1. The first occurrence is the initial analysis of the client and contract documentation. The second adoption of documentation analysis is for analysing the recorded decisions from the workshop to adjust and detail the user stories. Workshops are also to discuss high-level user stories with the client. Similar to the teams in P1, where the Analyst defines the user stories, the Product Owner in each team in P2 defines the features and user stories. This activity takes place after the roadmap has been defined. The roadmap is defined by the Product Manager using the technique of informal discussions with stakeholders such as Civil Engineers and Domain Experts. The user stories defined at the end of the clarification or PI planning phase are then ready to be refined by the team.

Analysis

A common technique used by all the explored teams is the Refinement session. Here, any questions and uncertainties about the user story are discussed. Also, both teams in P1 model the details of the user story using wireframes and technical designs such as UML diagrams. Similarly, P2T1 also models technical specifications for complex user stories and occasionally the UI specification if necessary. P1T1 and P1T2 make use of Clickable Prototypes to detail the functionality as well as a way to reuse the prototype for the frontend development. Both teams in P2 use a Research User Story to handle complex user stories. Informal communication is also used by each team in P2 with the Domain Expert to clarify the details of the user story.

Moreover, both teams in P1 have separate task(s) for design. P2T1 also performs design activities similar to the teams in P1, but only the high-level architecture is documented as C4 diagrams, as the team also uses Code Comments to

explain specific reasoning and examples in the code. In contrast, P2T2 does not use design tasks, this is mainly because they develop with scripts that do not have a UI, and they are still in a learning phase with regard to the structuring of the code.

Specification

User Stories are structured using the hierarchy of Epics, Features, and User Stories. These artefacts are stored in Azure DevOps for each of the explored teams and the user stories are specified using the role, action, and benefit format. Also, for user stories in P1, they are expanded with acceptance criteria and a functional and technical description, as well as a hyperlink to the documentation. Similarly, the Domain Experts in P2 attach documentation to the user stories where necessary, to explain a certain topic or provide domain knowledge.

Validation

In P1, the use of documentation artefacts such as the Functional Document and the Test Report makes it possible for internal and external stakeholders to validate requirements. Also, user stories are reviewed and approved by the customer. After the product increment has been deployed, the client tests and verifies the correct implementation of the user story and reports any defects. In contrast, P2 uses informal discussions at the Refinement and Sprint Planning to clarify questions and obtain missing information for the user story.

Furthermore, the implemented user story is then tested to ensure conformance with the original requirement. In all teams but P2T2, the tester writes the test scripts and tests the implemented user story according to those test cases, whereas there is neither a tester in P2T2 nor defined test cases.

Requirements Management

All the teams handle requirements volatility in a similar manner. In P1, a change request process is used to address the new requirements. The existing requirements are not changed, rather new requirements are introduced. If the new requirements are closely related to an original requirement from the clarification phase, that is expanded with more details but not changed. Change documentation artefacts are created which are then linked to a user story. P2 also adopts a similar approach but with different artefacts. A new user story is introduced, or in some cases, an existing user story is only changed if it is still opened. The team also ensure that the Validation Tests and Regression Tests Cases artefacts are updated using the DoD.

8.1.3 SRQ3: Which kinds of documentation artefacts are used for the various requirements engineering practices within the ASD approach?

We discuss the differences and similarities of documentation artefacts used for the RE activities within the teams.

The Analysis Report is used for requirements *elicitation* within both projects.

The high-level requirements and roadmap for the teams in P2 are outlined in the Project Planning, whereas P1 uses the Initial Contracts and Client Documentation as a starting point as well as the Meeting Minutes and Recordings of the workshops. Besides, the Issues artefact is used by P1 to document open questions to the client. In addition to the Analysis Report, other documentation artefacts are used by the teams for *analysis*. In P2T1 C4 diagrams are used, while P1 uses Detailed Design and External Interface Design. Moreover, Wireframes and Clickable Prototypes are used to design the UI in P1.

In all teams, requirements are *specified* using the hierarchy of Epics, Features, and User Stories which are planned using the Sprint Backlog. Also, the Product Backlog is used by each team, whereas P2 refers to it as the Team Backlog. For the high-level requirements, P2 uses the Program Backlog, while P1 uses Functional Scenarios. P1 also use additional artefacts such as Clickable Prototypes, User Interface Design to specify requirements. To specify requirements that were not part of the original scope, Change Requests, Changes, and Defects are used which are also the documentation artefacts used for *requirements management*.

Lastly, requirements are *validated* by testing in all teams. P1 uses the Test Plan and Test Report, whereas P2 uses the Validation Plan and Test Cases. At the end of each sprint, the teams validate the outcome of the implementation of the user stories. In P2, the Regression Test Report is used, whereas P1 uses the Sprint Review Report. P1 uses additional artefacts such as the Functional Document and the Clickable Prototype to compare the implementation to the specified requirements.

8.1.4 SRQ4: What is the current state of the identified documentation (from SRQ3) within the project?

We address this sub-question by using the taxonomy of documentation issues by Aghanjani et al. (2019) which are known as *Information Content (What)*, *Information Content (How)*, and *Process Related*.

Information Content (What)

For this aspect of documentation, we discuss the correctness, completeness, and up-to-dateness of the documentation.

Correctness of documentation is ensured using informal communication such as discussions during the Refinement sessions. Furthermore, P1 makes higher use of documentation templates than P2: templates and examples are part of C1M, and the method also prescribes specific roles for writing and updating user stories. User stories are used in both projects, and they are written mainly by the Analysts in P1, and by the Product Owner in P2. The use of templates has also drawbacks, as the templates easily become outdated. Additional artefacts such as DoR are hardly used in both projects due to time constraints.

About *completeness*, all the known requirements are documented for each team, with the main source of truth being the Backlog in Azure DevOps. User stories are not the only artefact used for requirements: in P1T2, the Analyst attaches other requirements to user stories via hyperlinks; in P2T1, a wiki collects central

information about the requirements, and the user stories are disjoint. Also, P2T1 faces difficulties in finding the right balance between documenting and making the code self-documented. The lack of a clear file structure is seen in P2T2 which is a hindrance to keeping documentation complete and up-to-date. In P1, keeping the documentation complete appears to be the responsibility of the assignee, rather than a shared team responsibility.

Keeping the documentation *up-to-date* is a challenge: in agile development, as per the agile manifesto, there is more focus on implementation and testing activities rather than on documentation. Unfortunately, this leads to documentation debt. Different techniques are used to overcome this difficulty. In P1, the four-eyes principle is used to review documentation, although it is not always effective as it depends on individual commitment and motivation. Also, the decision to update documentation is at the discretion of the assignee of the user story. Teams in P2 do not have documentation tasks defined. In both projects, the scope of documentation per user story is not clear and there is no formal check, nor a process to ensure the update of documentation is not forgotten.

Regarding the sufficiency of the documentation for extension and maintenance of the application, the teams in P1 are positive, perhaps because they have some time nearing the end of the release to update the documentation. In P2, the lack of a process leads to ad-hoc solutions: P2T1 focuses on documenting the code itself, while P2T2 is building a knowledge bank and is planning to organise workshops. Also, multi-lingual documentation in P1 is an additional obstacle that increases the effort and the challenge of keeping the versions aligned. Lastly, for all the teams but P2T2, the versioning of changes is managed automatically by the tools, while P2T2 faces challenges due to the manual nature of the task.

Information Content (How)

For this aspect of documentation, we discuss the maintainability, readability, usability, and usefulness of the documentation.

Concerning *maintainability*, the teams in P1 find it easy to add changes to the documentation, due to the organisation of documentation prescribed by C1M. However, identifying what other documentation artefacts need to be updated is not always straightforward. P1 addresses this by mentioning which documentation artefact needs to be updated in the user story, however, this activity is sometimes forgotten. Furthermore, it requires additional effort to add changes to the documentation within P2T2 due to the organisation of the scripts and the lack of a clear file structure. Efforts are made by the teams to address the complexity of updating large documentation artefacts: P2T1 ensures modular documentation, whereas P1T1 addresses this via informal communication.

Readability is ensured through feedback on the documentation. P1 obtains feedback on the user stories during the workshops and the approval process. P2T1 uses peer reviews whereas P2T2 informally discuss this using the Scrum events. Necessary documentation artefacts for requirements validation are not reviewed until nearing the end of the release when it is rapidly reviewed. For all the teams, it is difficult to determine the conciseness of the contents of the documentation without adequate feedback from external and internal stakeholders.

Issues faced with the tools affect the *usability* of the documentation. Teams in P1 face connectivity issues with SharePoint. Receiving feedback from external stakeholders via email requires a manual effort to update the documentation artefact either in SharePoint or Azure DevOps. Also, P2T1 faces limitations when updating the C4 diagrams in PlantUML due to its level of expressiveness. So far, the teams in P1 have not received any questions about the documentation from external stakeholders. P2T2 receives questions from external stakeholders and is building a knowledge bank to address the questions. Due to the functionality-based division of teams in each project, documentation is shared on a higher level: P2 shares documentation on a feature and program increment level, while P1 shares a high-level Functional Documentation artefact. This artefact is updated by one assignee to avoid conflicting changes. Although there are alignment meetings within P1, they hardly incorporate a review of documentation.

We evaluate *usefulness* by feedback given from stakeholders. The process of encouraging feedback is absent within the teams. P1 lacks this process in the DoD, whereas in P2T1, receiving feedback depends on the type of artefact and the stakeholder. Neither of the teams has a process in place to address the incorporation of feedback to the documentation, which has led to ad-hoc solutions. The teams discuss this via the Daily Scrum. Additionally, P2 addresses the comments on high-level requirements by means of discussions among the Product Owners.

Process Related

To address the strengths and challenges related to the process of writing documentation, we address internationalisation, contribution, and automated documentation issues. The *internationalisation* of the teams in P1 requires additional effort to synchronise the versions of documentation artefacts. To address the language barrier, documentation artefacts intended for internal use are kept in English as well as the language used in the Scrum events.

Contribution to documentation depends on the planning of the documentation. Neither of the teams has a process to ensure the planning of documentation, mainly due to the agile mindset of prioritising the working software and documenting less. Also, due to time constraints documentation tends to be scrapped within the teams. P1 updates the missing documentation nearing the end of the release, whereas in P2 it is difficult to address the missing documentation, resulting in a huge documentation debt. Furthermore, none of the four teams has a process to report issues found in the documentation, this is addressed by means of informal communication. Also, the lack of awareness about the relevance of documentation in P2T1 led to a suggestion to increase the awareness of the value of documenting within the team.

There are a few *automated documentation* in the explored teams. P1 uses HTML clickable prototypes and unit tests, whereas P2T1 uses C4 diagrams and unit tests. P2T2 does not have any automated documentation. It is difficult to automate the documentation within P2 due to the pre-required domain knowledge.

We identify the *strengths* of documentation processes. In P1, the template and review of user stories ensure that each user story is of high quality and includes

a documentation task. Also, the team manager enforces the completion of documentation tasks. Also, the process of updating documentation is accelerated due to the availability of templates and examples prescribed by C1M. In P2T1, the awareness of documentation has increased over the years, whereas P2T2 is developing a knowledge-sharing platform.

8.1.5 SRQ5: What is the impact of requirements change on the identified documentation (from SRQ3) within the project?

The change of requirements is inevitable for the explored projects. SRQ2 and SRQ4 already discussed aspects of this topic, however, this sub-research question zooms into the challenges of incorporating changing requirements in the documentation. In P1, a number of challenges are faced. Although there is a formal process in place to handle the change requests, the documentation aspect is not enforced. The documentation tends to be skipped as the change usually entails a small functionality. As a result, the skipped documentation piles up into a huge documentation debt. Also, the limitations of the tools and the lack of clarity on the scope of documentation hinder the process of updating documentation.

In P2T2 using Code Comments to keep track of the changes to the code, introduces additional overhead to incorporate change in the project. After new user stories are introduced and implemented, the references and versioning have to be updated manually using code comments. Due to the poor structuring of documentation this activity requires additional effort. However, P2T1 address change by ensuring their documentation is kept modular, thereby requiring less effort to update the documentation.

8.1.6 SRQ6: What framework can be used to address documentation issues by improving documentation practices in multi-team software projects?

From the problem investigation phase, we identified issues such as the lack of planning and estimation, time constraints, lack of a clear scope, and the lack of review for documentation. We observed variations in the organisation and structure of scaled-ASD projects. The process maturity level of the adoption of ASD varies even within the explored projects. Hence, the need for a lightweight and easily adaptable framework.

Furthermore, in self-organising teams, a value of ASD is constant improvement from within the team rather than imposing external guidelines. The literature emphasises the benefits of informal communication within feedback cycles to improve processes within Agile teams. Therefore, we combine the strengths of informal communication within agile teams with the recommendations from the literature to design a framework that addresses the issues found in the documentation.

We designed and propose a lightweight framework known as the DIFFC model consisting of Planning, Monitoring, and Reflecting phases. The Planning phase

ensures that documentation is not forgotten as part of user stories. The Monitoring Phase ensures that the progress of the documentation, as well as any dependencies, are reported. Lastly, the Reflecting phase allows the team to address the issues concerning documentation and improve the documentation processes within the team. This phase combines a data-driven approach and a game-based approach with the pre-existing discussion of perceptions within the team. The output of this phase is process improvement actions for the next sprint to improve aspects of the documentation.

8.1.7 SRQ7: What is the effectiveness of the proposed framework from SRQ6?

We studied the effectiveness of the DIFFC model through an experiment and concluded with a validation survey. The experiment was conducted using two cases: an in-depth study was conducted with P3, whereas a preliminary study was conducted in P2 due to timing and resource constraints. Reusing the variables used in the experiment of Green et al. (2005), we applied the DIFFC model to the Retrospective, Sprint Planning, and Daily Scrum in an experiment to observe the Ease of use, Quality, Productivity, Usefulness, and Use of the model.

Although the experiment was only conducted for one sprint due, the results are promising. Considering the *ease of use* of the model, the guidelines were clear and easy to understand. During the observation of the Reflecting phase, there were questions in the beginning, but afterwards, the team gained familiarity with using the tooling and guidelines. The *productivity* increased with regard to keeping the documentation up-to-date by the introduction of documentation tasks for the Change Requests in P3 with estimations and clarified scope. Also, awareness of the *quality* and the current state of documentation increased within all the teams. In general, there were a number of Process Improvement actions collected from the ideas, even though it is not as much as expected. There was positive feedback from the teams with regard to the *usefulness* of the model: it helped create awareness and interactively involved the team to collect ideas about how the team can improve the challenges facing documentation. Due to the early nature of the results, we cannot generalise that the model will be *used*, however, the results are promising with potential areas to improve in the DIFFC model.

8.2 Implications for Research and Practitioners

Our main aim was to explore the impact of RE practices in scaled-ASD on documentation and propose a framework to address the issues found in the documentation. We have done so by conducting case studies with semi-structured interviews, documentation analysis, designing a treatment and experimenting with the proposed treatment and a real-world context.

We contributed to the literature by adding evidence of the variations in the adoption of scaled-ASD methods from the multiple-case studies. We provide empirical data on the RE activities found on the scaled-ASD approach of each project with the accompanying documentation artefacts. The issues of mini-

mal documentation and requirements volatility in ASD were tackled using the taxonomy of documentation issues by Aghanjani et al. (2019). The challenges facing documentation confirmed the literature with regard to the agile way of working with more focus on implementation and testing activities and less focus on documenting. Our findings contribute to the literature by proposing a framework to address the challenges using the existing processes of agile teams.

Furthermore, we provide practitioners in the software engineering industry with insights into the various similarities and differences in the adoption of ASD in two multi-team software projects. Projects can learn from each other with regard to the shared solutions to the challenges facing the documentation. More importantly, the DIFFC model is valuable in addressing the issues found in the documentation and improving this process as a team. Even though the results are preliminary, they are promising and the model has the potential to address the renowned issue of documentation in ASD.

8.3 Implications for Future Work

The exploratory nature of our research raises a number of opportunities for future research in terms of theory development and treatment validation.

Firstly, while we scoped our research to scaled-ASD projects, we believe that if the context is expanded to include smaller projects or single-team projects, the results will be comparable. This assumption is based on the findings that even though the explored projects are large, the teams are divided based on functionality and only interact on a higher level. Therefore, it will be interesting for future research to apply this research protocol and the proposed DIFFC model to other ASD projects in order to generalise the results.

Moreover, the scope of the validation experiment could be much broader as we were limited by time and resource constraints. We recommend future research to study the effectiveness by observing longer periods, ideally more than three sprints, as we saw that the first Retrospective was mainly to create awareness and the second derived more process improvement actions than the first. In addition to the in-depth observation, we propose future research to include more projects in the validation of effectiveness with a variety of scaled-ASD approaches.

Lastly, we observed two main aspects for improvements regarding the DIFFC model: The preparation of data and a subject for discussion among the team during the Retrospectives; The completion of the documentation tasks, which were added during the Planning phase. We recommend an action research approach to improve the model with regard to these aspects and investigate the impact. The latter relates to the motivation of the team members toward updating documentation which is out of the scope of the proposed model. The subject of motivation is certainly a strong recommendation for future research.

Chapter 9

Discussions

We discuss the limitations of this research. We explain for each of the four categories the threats to validity: construct validity, internal validity, external validity, and reliability [22]; as well as how we mitigate these threats.

9.1 Research Limitations

Construct validity ensures that the correct operational measures are identified for the concept being studied [22]. The goal to satisfy construct validity is to make research as objective as possible. We mitigate this threat during the data collection phase by having multiple sources of evidence. We collect data from semi-structured interviews and documentation analysis with two cases, and multiple people in one team to eliminate the subjectivity in data as much as possible. However, since not one agile method suits every organisation, we expect to find subjectivity in the way the agile methods are presented by the participants. For the validation research, we incorporate two sources of data an experiment and a validation survey with two cases. Also, we mitigate the threat of construct validity by making a report of each case study as well as an experimental report. However, due to the agreed confidentiality, these sources are not openly available.

Furthermore, *internal validity* seeks to establish a causal relationship for explanatory case studies [22]. Another concern of internal validity is making inferences [22]. We mitigate this threat using the suggested tactics by Yin (2018) such as pattern matching, addressing rival explanations, and using logic models [22]. An example of how we mitigate this threat is by modelling the conceptual model in Figure 3.9 and using it as a guide to collect the data. We address rival explanations in the perceptions of the current state of documentation using interviews with multiple team members and documentation analysis as an inspection. The case study protocol and the experimental guidelines ensure that we use themes to guide the study with a logical flow. These were reviewed by means of a four-eyes principle with the first supervisor. Additionally, in the validation experiment, we compare our observations to the results of the survey which is based on the same variables. This gives us a clear insight

to determine the effectiveness of our proposed model. However, we admit that the pre-post comparison data is subjected to limitations as it was collected from the Scrum Masters. This threat could not be mitigated as the researcher did not have direct access to the environments of those projects, due to the privacy policies of the projects. Hence, we ask for an explanation of the data from the Scrum Masters to ensure the credibility of these figures.

External validity addresses whether the findings from the case study can be generalised outside the context of the investigated cases [22]. Here, we recognise a potential bias in our results since all the samples are collected using purposive sampling at large-scale ASD projects. Our attempt to mitigate this threat is by selecting projects from a variety of organisations in various sectors. A total of two projects and selection criteria defined in Section 3.4 ensure that the data may lead to generalisable conclusions for multi-team software projects. Additionally, the scope of our research is limited to that of RE-related documentation since the subject of documentation is broad. Here, we accept that the findings based on RE-related documentation might be different from other kinds of documentation, but other kinds of documentation are out of the scope of this research. Moreover, we admit that due to timing limitations, our results from the validation study are preliminary and require a much longer period of observations rather than one four-week sprint to draw solid conclusions.

Finally, *reliability validity* is concerned with whether the operations of the study, such as the data collection procedures, can be repeated with the same results [22]. Yin (2018) recommended some tactics to mitigate the threat of reliability which are developing a case study database and maintaining a chain of evidence [22]. Therein, we log the chain of evidence as discussed in construct validity, but due to the confidentiality of the interviews and project-specific information, the consensual agreement prohibits the use of the collected data outside the purpose of this study. Therefore, there is no database available for the public audience. However, the interview questions and experimental guidelines are included as Appendices in this thesis.

Bibliography

[1] C. J. Stettina and W. Heijstek, "Necessary and neglected? An empirical study of internal documentation in agile software development teams," in *Proceedings of the 29th ACM international conference on Design of communication*, 2011, pp. 159–166.

[2] B. Hobbs and Y. Petit, "Agile methods on large projects in large organizations," *Project Management Journal*, vol. 48, no. 3, pp. 3–19, 2017.

[3] K. N. Rao, G. K. Naidu, and P. Chakka, "A study of the agile software development methods, applicability and implications in industry," *International Journal of Software Engineering and its applications*, vol. 5, no. 2, pp. 35–45, 2011.

[4] M. Al-Zewairi, M. Biltawi, W. Etaiwi, A. Shaout, *et al.*, "Agile software development methodologies: Survey of surveys," *Journal of Computer and Communications*, vol. 5, no. 05, p. 74, 2017.

[5] K. Beck, M. Beedle, A. Van Bennekum, *et al.*, *The agile manifesto.* 2001.

[6] S. W. Ambler, "The agile scaling model (ASM): Adapting agile methods for complex environments," pp. 1–35, 2009.

[7] P. Kettunen, "Extending software project agility with new product development enterprise agility," *Software Process: Improvement and Practice*, vol. 12, no. 6, pp. 541–548, 2007.

[8] T. Dingsøyr, S. Nerur, V. Balijepally, and N. B. Moe, "A decade of agile methodologies: Towards explaining agile software development," *Journal of systems and software*, vol. 85, no. 6, pp. 1213–1221, 2012.

[9] M. Lindvall, D. Muthig, A. Dagnino, *et al.*, "Agile software development in large organizations," *Computer*, vol. 37, no. 12, pp. 26–34, 2004.

[10] J. B. Barlow, J. Giboney, M. J. Keith, *et al.*, "Overview and guidance on agile development in large organizations," *Communications of the Association for Information Systems*, vol. 29, no. 2, pp. 25–44, 2011.

[11] A. De Lucia and A. Qusef, "Requirements engineering in agile software development," *Journal of emerging technologies in web intelligence*, vol. 2, no. 3, pp. 212–220, 2010.

[12] T. V. Ribeiro, C. D. F. Souza, and H. A. T. Leao, "Sidd-scrum iteration driven development: An agile software development and management process based on scrum (s).," in *SEKE*, 2018, pp. 502–501.

[13] E. Bjarnason, K. Wnuk, and B. Regnell, "A case study on benefits and side-effects of agile practices in large-scale requirements engineering," in *proceedings of the 1st workshop on agile requirements engineering*, 2011, pp. 1–5.

[14] S. C. B. de Souza, N. Anquetil, and K. M. de Oliveira, "A study of the documentation essential to software maintenance," in *Proceedings of the 23rd annual international conference on Design of communication: documenting & designing for pervasive information*, 2005, pp. 68–75.

[15] T. Clear, "Documentation and agile methods: Striking a balance," *ACM SIGCSE Bulletin*, vol. 35, no. 2, pp. 12–13, 2003.

[16] C. J. Stettina, W. Heijstek, and T. E. Fægri, "Documentation work in agile teams: The role of documentation formalism in achieving a sustainable practice," in *2012 Agile Conference*, IEEE, 2012, pp. 31–40.

[17] T. Theunissen, U. van Heesch, and P. Avgeriou, "A mapping study on documentation in continuous software development," *Information and Software Technology*, vol. 142, p. 106 733, 2022.

[18] G. Garousi, V. Garousi-Yusifoğlu, G. Ruhe, J. Zhi, M. Moussavi, and B. Smith, "Usage and usefulness of technical software documentation: An industrial case study," *Information and Software Technology*, vol. 57, pp. 664–682, 2015.

[19] B. Curtis, H. Krasner, and N. Iscoe, "A field study of the software design process for large systems," *Communications of the ACM*, vol. 31, no. 11, pp. 1268–1287, 1988.

[20] E. Aghajani, C. Nagy, O. L. Vega-Márquez, *et al.*, "Software documentation issues unveiled," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, IEEE, 2019, pp. 1199–1210.

[21] R. J. Wieringa, *Design science methodology for information systems and software engineering*. Springer, 2014.

[22] R. K. Yin, *Case study research and applications - design and methods*. Sage Publications, 2018.

[23] A. Chakraborty, M. K. Baowaly, A. Arefin, and A. N. Bahar, "The role of requirement engineering in software development life cycle," *Journal of emerging trends in computing and information sciences*, vol. 3, no. 5, pp. 723–729, 2012.

[24] I. Inayat, S. S. Salim, S. Marczak, M. Daneva, and S. Shamshirband, "A systematic literature review on agile requirements engineering practices and challenges," *Computers in human behavior*, vol. 51, pp. 915–929, 2015.

[25] G. Wagenaar, S. Overbeek, G. Lucassen, S. Brinkkemper, and K. Schneider, "Working software over comprehensive documentation–rationales of agile teams for artefacts usage," *Journal of Software Engineering Research and Development*, vol. 6, no. 1, pp. 1–23, 2018.

[26] K. Madampe, R. Hoda, J. Grundy, and P. Singh, "Towards understanding technical responses to requirements changes in agile teams," in *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, 2020, pp. 153–156.

[27] G. Wong, T. Greenhalgh, G. Westhorp, J. Buckingham, and R. Pawson, "Rameses publication standards: Meta-narrative reviews," *Journal of Advanced Nursing*, vol. 69, no. 5, pp. 987–1004, 2013.

[28] H. Edison, X. Wang, and K. Conboy, "Comparing methods for large-scale agile software development: A systematic literature review," *IEEE Transactions on Software Engineering*, 2021.

[29] R. Hoda, N. Salleh, J. Grundy, and H. M. Tee, "Systematic literature reviews in agile software development: A tertiary study," *Information and software technology*, vol. 85, pp. 60–70, 2017.

[30] A. Mishra and D. Dubey, "A comparative study of different software development life cycle models in different scenarios," *International Journal of Advance Research in Computer Science and Management Studies*, vol. 1, no. 5, pp. 64–69, 2013.

[31] N. B. Ruparelia, "Software development lifecycle models," *ACM SIGSOFT Software Engineering Notes*, vol. 35, no. 3, pp. 8–13, 2010.

[32] R. Ibrahim and S. Y. Yey, "Formalization of the data flow diagram rules for consistency check," *International Journal of Software Engineering Applications*, vol. 1, no. 4, pp. 95–111, 2010. DOI: 10.5121/ijsea.2010.1406.

[33] J. Henkel, C. Bird, S. K. Lahiri, and T. Reps, "Learning from, understanding, and supporting devops artifacts for docker," in *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, IEEE, 2020, pp. 38–49.

[34] P. Perera, R. Silva, and I. Perera, "Improve software quality through practicing devops," in *2017 Seventeenth International Conference on Advances in ICT for Emerging Regions (ICTer)*, IEEE, 2017, pp. 1–6.

[35] A. Alshamrani and A. Bahattab, "A comparison between three sdlc models waterfall model, spiral model, and incremental/iterative model," *International Journal of Computer Science Issues (IJCSI)*, vol. 12, no. 1, p. 106, 2015.

[36] K. Jammalamadaka and V. R. Krishna, "Agile software development and challenges," *International Journal of Research in Engineering and Technology*, vol. 2, no. 08, pp. 125–129, 2013.

[37] S. Shylesh, "A study of software development life cycle process models," in *National Conference on Reinventing Opportunities in Management, IT, and Social Sciences*, 2017, pp. 534–541.

[38] S. Nerur, R. Mahapatra, and G. Mangalaraj, "Challenges of migrating to agile methodologies," *Communications of the ACM*, vol. 48, no. 5, pp. 72–78, 2005.

[39] P. Abrahamsson, J. Warsta, M. T. Siponen, and J. Ronkainen, "New directions on agile methods: A comparative analysis," in *25th International Conference on Software Engineering, 2003. Proceedings.*, Ieee, 2003, pp. 244–254.

[40] M. R. J. Qureshi and J. S. Ikram, "Proposal of enhanced extreme programming model," *International Journal of Information Engineering and Electronic Business*, vol. 7, no. 1, p. 37, 2015.

[41] R. Fojtik, "Extreme programming in development of specific software," *Procedia Computer Science*, vol. 3, pp. 1464–1468, 2011.

[42] G. S. Matharu, A. Mishra, H. Singh, and P. Upadhyay, "Empirical study of agile software development methodologies: A comparative analysis," *ACM SIGSOFT Software Engineering Notes*, vol. 40, no. 1, pp. 1–6, 2015.

[43] K. Schwaber and J. Sutherland, "The scrum guide," *Scrum Alliance*, vol. 21, no. 19, p. 1, 2011.

[44] D. J. Anderson, *Kanban: successful evolutionary change for your technology business*. Blue Hole Press, 2010.

[45] R. B. Wakode, L. P. Raut, and P. Talmale, "Overview on kanban methodology and its implementation," *IJSRD-International Journal for Scientific Research & Development*, vol. 3, no. 02, pp. 2321–0613, 2015.

[46] M. O. Ahmad, J. Markkula, and M. Oivo, "Kanban in software development: A systematic literature review," in *2013 39th Euromicro conference on software engineering and advanced applications*, IEEE, 2013, pp. 9–16.

[47] M. O. Ahmad, P. Kuvaja, M. Oivo, and J. Markkula, "Transition of software maintenance teams from scrum to kanban," in *2016 49th Hawaii International Conference on System Sciences (HICSS)*, IEEE, 2016, pp. 5427–5436.

[48] C. Matthies, "Agile process improvement in retrospectives," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, IEEE, 2019, pp. 150–152.

[49] B. Nuseibeh and S. Easterbrook, "Requirements engineering: A roadmap," in *Proceedings of the Conference on the Future of Software Engineering*, 2000, pp. 35–46.

[50] O. Dieste Tubio, M. Lopez, and F. Ramos, "Updating a systematic review about selection of software requirements elicitation techniques," 2009.

[51] D. Zowghi and C. Coulin, "Requirements elicitation: A survey of techniques, approaches, and tools," in *Engineering and managing software requirements*, Springer, 2005, pp. 19–46.

[52] L. Zamudio, J. A. Aguilar, C. Tripp, and S. Misra, "A requirements engineering techniques review in agile software development methods," in *International Conference on Computational Science and Its Applications*, Springer, 2017, pp. 683–698.

[53] L. Bass, J. Bergey, P. Clements, P. Merson, I. Ozkaya, and R. Sangwan, "A comparison of requirements specification methods from a software architecture perspective," Tech. Rep., 2006.

[54] J. Estublier, "Software configuration management: A roadmap," in *Proceedings of the Conference on the Future of Software Engineering*, 2000, pp. 279–289.

[55] B. Ramesh, L. Cao, and R. Baskerville, "Agile requirements engineering practices and challenges: An empirical study," *Information Systems Journal*, vol. 20, no. 5, pp. 449–480, 2010.

[56] L. Cao and B. Ramesh, "Agile requirements engineering practices: An empirical study," *IEEE software*, vol. 25, no. 1, pp. 60–67, 2008.

[57] A. Eberlein and J. Leite, "Agile requirements definition: A view from requirements engineering," in *Proceedings of the International Workshop on Time-Constrained Requirements Engineering (TCRE'02)*, 2002, pp. 4–8.

[58] S. Heng, "Impact of unified user-story-based modeling on agile methods: Aspects on requirements, design and life cycle management," Ph.D. dissertation, Université catholique de Louvain, Louvain La Neuve, Belgique, 2017.

[59] M. Daneva, E. Van Der Veen, C. Amrit, *et al.*, "Agile requirements prioritization in large-scale outsourced system projects: An empirical study," *Journal of systems and software*, vol. 86, no. 5, pp. 1333–1353, 2013.

[60] K. Boness and R. Harrison, "Goal sketching: Towards agile requirements engineering," in *International Conference on Software Engineering Advances (ICSEA 2007)*, IEEE, 2007, pp. 71–71.

[61] N. A. Ernst, A. Borgida, I. J. Jureta, and J. Mylopoulos, "Agile requirements engineering via paraconsistent reasoning," *Information systems*, vol. 43, pp. 100–116, 2014.

[62] D. M. Berry, "The inevitable pain of software development, including of extreme programming, caused by requirements volatility," 2002.

[63] N. N. B. Abdullah, S. Honiden, H. Sharp, B. Nuseibeh, and D. Notkin, "Communication patterns of agile requirements engineering," in *Proceedings of the 1st workshop on agile requirements engineering*, 2011, pp. 1–4.

[64] Y. Yu and H. Sharp, "Analysing requirements in a case study of pairing," in *Proceedings of the 1st Workshop on Agile Requirements Engineering*, 2011, pp. 1–6.

[65] Y. Zhu, *Requirements engineering in an agile environment*, 2009.

[66] T. Dingsøyr, N. B. Moe, T. E. Fægri, and E. A. Seim, "Exploring software development at the very large-scale: A revelatory case study and research agenda for agile method adaptation," *Empirical Software Engineering*, vol. 23, no. 1, pp. 490–520, 2018.

[67] D. J. Reifer, F. Maurer, and H. Erdogmus, "Scaling agile methods," *IEEE software*, vol. 20, no. 4, pp. 12–14, 2003.

[68] T. Dybå and T. Dingsøyr, "Empirical studies of agile software development: A systematic review," *Information and software technology*, vol. 50, no. 9-10, pp. 833–859, 2008.

[69] M. Laitinen, M. E. Fayad, and R. P. Ward, "Thinking objectively: The problem with scalability," *Communications of the ACM*, vol. 43, no. 9, pp. 105–107, 2000.

[70] D. Leffingwell, *Scaling software agility: best practices for large enterprises*. Pearson Education, 2007.

[71] C. Larman and B. Vodde, *Large-scale scrum: More with LeSS*. Addison-Wesley Professional, 2016.

[72] B. Aschauer, P. Hruschka, K. Lauenroth, M. Meuten, and G. Rogers, *Handbook of RE@Agile According to the IREB Standard*. International Requirements Engineering Board, 2019.

[73] A. Putta, Ö. Uludağ, S.-L. Hong, M. Paasivaara, and C. Lassenius, "Why do organizations adopt agile scaling frameworks? a survey of practitioners," in *Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2021, pp. 1–12.

[74] D. Leffingwell, *Agile software requirements: lean requirements practices for teams, programs, and the enterprise*. Addison-Wesley Professional, 2010.

[75] K. Schwaber and M. Beedle, *Agile software development with Scrum*. Prentice Hall Upper Saddle River, 2002, vol. 1.

[76] K. Beck, *Extreme programming explained: embrace change*. addison-wesley professional, 2000.

[77] A. Vaidya, "Does dad know best, is it better to do less or just be safe? adapting scaling agile practices into the enterprise," *Pacific NW Software Quality Conference (PNSQC)*, pp. 1–18, 2014.

[78] M. Kuhrmann, P. Diebold, J. Munch, *et al.*, "Hybrid software development approaches in practice: A european perspective," *IEEE software*, vol. 36, no. 4, pp. 20–31, 2018.

[79] M. Kuhrmann, J. Münch, P. Diebold, O. Linssen, and C. Prause, *On the use of hybrid development approaches in software and systems development: construction and test of the HELENA survey*. Gesellschaft für Informatik, 2016.

[80] D. Turk, F. Robert, and B. Rumpe, "Assumptions underlying agile software-development processes," *Journal of Database Management (JDM)*, vol. 16, no. 4, pp. 62–87, 2005.

[81] A. Forward and T. C. Lethbridge, "The relevance of software documentation, tools and technologies: A survey," in *Proceedings of the 2002 ACM symposium on Document engineering*, 2002, pp. 26–33.

[82] J. Zhi, V. Garousi-Yusifoğlu, B. Sun, G. Garousi, S. Shahnewaz, and G. Ruhe, "Cost, benefits and quality of software development documentation: A systematic mapping," *Journal of Systems and Software*, vol. 99, pp. 175–198, 2015.

[83] Y. Shmerlin, I. Hadar, D. Kliger, and H. Makabee, "To document or not to document? an exploratory study on developers' motivation to document code," in *International Conference on Advanced Information Systems Engineering*, Springer, 2015, pp. 100–106.

[84] T. T. Barker, *Writing software documentation: A Task-oriented Approach*. Allyn and Bacon, 2003, vol. 2.

[85] W. Ding, P. Liang, A. Tang, and H. Van Vliet, "Knowledge-based approaches in software documentation: A systematic literature review," *Information and Software Technology*, vol. 56, no. 6, pp. 545–567, 2014.

[86] L. C. Briand, "Software documentation: How much is enough?" In *Seventh European Conference on Software Maintenance and Reengineering, 2003. Proceedings.*, IEEE, 2003, pp. 13–15.

[87] I. Sommerville, "Integrated requirements engineering: A tutorial," *IEEE software*, vol. 22, no. 1, pp. 16–23, 2005.

[88] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical software engineering*, vol. 14, no. 2, pp. 131–164, 2009.

[89] T. C. Lethbridge, J. Singer, and A. Forward, "How software engineers use documentation: The state of the practice," *IEEE software*, vol. 20, no. 6, pp. 35–39, 2003.

[90] E. Arisholm, L. C. Briand, S. E. Hove, and Y. Labiche, "The impact of UML documentation on software maintenance: An experimental evaluation," *IEEE Transactions on Software Engineering*, vol. 32, no. 6, pp. 365–381, 2006.

[91] G. Wagenaar, R. Helms, D. Damian, and S. Brinkkemper, "Artefacts in agile software development," in *International Conference on Product-Focused Software Process Improvement*, Springer, 2015, pp. 133–148.

[92] G. Wagenaar, S. Overbeek, G. Lucassen, S. Brinkkemper, and K. Schneider, "Influence of software product management maturity on usage of artefacts in agile software development," in *International Conference on Product-Focused Software Process Improvement*, Springer, 2017, pp. 19–27.

[93] M. Kuhrmann, D. M. Fernández, and M. Gröber, "Towards artifact models as process interfaces in distributed software projects," in *2013 IEEE 8th International Conference on Global Software Engineering*, IEEE, 2013, pp. 11–20.

[94] W. Gerard, S. Overbeek, and S. Brinkkemper, "Fuzzy artefacts: Formality of communication in agile teams," in *2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC)*, IEEE, 2018, pp. 1–7.

[95] H. Sharp, H. Robinson, and M. Petre, "The role of physical artefacts in agile software development: Two complementary perspectives," *Interacting with computers*, vol. 21, no. 1-2, pp. 108–116, 2009.

[96] M. Alqudah and R. Razali, "A review of scaling agile methods in large software development," *International Journal on Advanced Science, Engineering and Information Technology*, vol. 6, no. 6, pp. 828–837, 2016.

[97] S. Ambler, *Agile modeling: effective practices for extreme programming and the unified process*. John Wiley & Sons, 2002.

[98] V. R. Basili, G. Caldiera, and H. D. Rombach, "The goal question metric approach," *Encyclopedia of software engineering*, pp. 528–532, 1994.

[99] A. Fuggetta, L. Lavazza, S. Morasca, S. Cinti, G. Oldano, and E. Orazi, "Applying gqm in an industrial software factory," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 7, no. 4, pp. 411–448, 1998.

[100] T. de Souza Machado, Y. T. Ximenes, V. de Oliveira Neves, and L. C. de Castro Salgado, "A case study on the perceptions of IT professionals during the transition from a traditional to an agile process model," in *Anais do XVII Simpósio Brasileiro de Sistemas de Informação*, SBC, 2021.

[101] G. C. Green, A. R. Hevner, and R. W. Collins, "The impacts of quality and productivity perceptions on the use of software process improvement innovations," *Information and Software Technology*, vol. 47, no. 8, pp. 543–553, 2005.

[102] T. Dingsøyr, T. E. Fægri, and J. Itkonen, "What is large in large-scale? a taxonomy of scale for agile software development," in *International Conference on Product-Focused Software Process Improvement*, Springer, 2014, pp. 273–276.

[103] G. Gutierrez, J. Garzas, M. T. G. de Lena, and J. M. Moguerza, "Self-managing: An empirical study of the practice in agile teams," *IEEE Software*, vol. 36, no. 1, pp. 23–27, 2018.

[104] C. Sibona and S. Walczak, "Purposive sampling on twitter: A case study," in *2012 45th Hawaii International Conference on System Sciences*, IEEE, 2012, pp. 3510–3519.

[105] M. Marshall, "The key informant technique," *Family Practice*, vol. 13, no. 1, pp. 92–97, Feb. 1996, ISSN: 0263-2136.

[106] T. C. Lethbridge, S. E. Sim, and J. Singer, "Studying software engineers: Data collection techniques for software field studies," *Empirical software engineering*, vol. 10, no. 3, pp. 311–341, 2005.

[107] D. S. Cruzes and T. Dyba, "Recommended steps for thematic synthesis in software engineering," in *2011 international symposium on empirical software engineering and measurement*, IEEE, 2011, pp. 275–284.

[108] A. Hess, P. Diebold, and N. Seyff, "Understanding information needs of agile teams to improve requirements communication," *Journal of Industrial Information Integration*, vol. 14, pp. 3–15, 2019.

[109] A. J. Shenhar, D. Dvir, T. Lechler, and M. Poli, "One size does not fit all: True for projects, true for frameworks," in *Proceedings of PMI research conference*, Project Management Institute, 2002, pp. 14–17.

[110] M. C. Paulk, B. Curtis, M. B. Chrissis, and C. V. Weber, "Capability maturity model, version 1.1," *IEEE software*, vol. 10, no. 4, pp. 18–27, 1993.

[111] D. N. Card, "Research directions in software process improvement," in *Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004. COMPSAC 2004.*, IEEE, 2004, 238–vol.

[112] C. Santana, F. Queiroz, A. Vasconcelos, and C. Gusmão, "Software process improvement in agile software development a systematic literature review," in *2015 41st Euromicro Conference on Software Engineering and Advanced Applications*, IEEE, 2015, pp. 325–332.

[113] O. Salo, "Enabling software process improvement in agile software development teams and organisations," PhD dissertation, University of Oulu, Finland, 2006.

[114] T. Dingsøyr and T. Dybå, "Team effectiveness in software development: Human and cooperative aspects in team effectiveness models and priorities for future studies," in *2012 5th international workshop on cooperative and human aspects of software engineering (chase)*, IEEE, 2012, pp. 27–29.

[115] K. S. Rubin, *Essential Scrum: A practical guide to the most popular Agile process*. Addison-Wesley, 2012.

[116] M. Zarour, A. Abran, J.-M. Desharnais, and A. Alarifi, "An investigation into the best practices for the successful design and implementation of lightweight software process assessment methods: A systematic literature review," *Journal of Systems and Software*, vol. 101, pp. 180–192, 2015.

[117] M. Hummel, C. Rosenkranz, and R. Holten, "The role of communication in agile systems development," *Business & Information Systems Engineering*, vol. 5, no. 5, pp. 343–355, 2013.

[118] H. Karhatsu, M. Ikonen, P. Kettunen, F. Fagerholm, and P. Abrahamsson, "Building blocks for self-organizing software development teams a framework model and empirical pilot study," in *2010 2nd International Conference on Software Technology and Engineering*, IEEE, vol. 1, 2010, pp. 297–304.

[119] V. Stray, N. Moe, and A. Aurum, "Investigating daily team meetings in agile software projects," in *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*, Los Alamitos, CA, USA: IEEE Computer Society, 2012, pp. 274–281. DOI: 10.1109/SEAA.2012.16.

[120] V. Stray, N. B. Moe, and D. I. Sjoberg, "Daily stand-up meetings: Start breaking the rules," *IEEE Software*, vol. 37, no. 03, pp. 70–77, 2018.

[121] C. Matthies, "Playing with your project data in scrum retrospectives," in *2020 IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, IEEE, 2020, pp. 113–115.

[122] R. E. Kraut and L. A. Streeter, "Coordination in software development," *Communications of the ACM*, vol. 38, no. 3, pp. 69–81, 1995, ISSN: 0001-0782. DOI: 10.1145/203330.203345.

[123] S. Kauffeld and N. Lehmann-Willenbrock, "Meetings matter: Effects of team meetings on team and organizational success," *Small group research*, vol. 43, no. 2, pp. 130–158, 2012.

[124] M. Pikkarainen, J. Haikara, O. Salo, P. Abrahamsson, and J. Still, "The impact of agile practices on communication in software development," *Empirical Software Engineering*, vol. 13, no. 3, pp. 303–337, 2008.

[125] P. Caroli and T. C. Coimbra, *Funretrospectives: Activities and Ideas for Making Agile Retrospectives More Engaging*. Leanpub, Layton, 2015.

[126] M. Jovanović, A.-L. Mesquida, N. Radaković, and A. Mas, "Agile retrospective games for different team development phases," *Journal of Universal Computer Science*, vol. 22, no. 12, pp. 1489–1508, 2016.

[127] C. Ziftci and J. Reardon, "Who broke the build? automatically identifying changes that induce test failures in continuous integration at google scale," in *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, IEEE, 2017, pp. 113–122.

[128] E. Derby, D. Larsen, and K. Schwaber, *Agile retrospectives: Making good teams great*. Pragmatic Bookshelf, 2006.

[129] C. R. Pandian, *Software metrics: A guide to planning, analysis, and application*. Auerbach Publications, 2003.

[130] K. M. Bumbary, "Using velocity, acceleration, and jerk to manage agile schedule risk," in *2016 International Conference on Information Systems Engineering (ICISE)*, IEEE, 2016, pp. 73–80.

[131] H. Sedehi and G. Martano, "Metrics to evaluate & monitor agile based software development projects-a fuzzy logic approach," in *2012 Joint Conference of the 22nd International Workshop on Software Measurement and the 2012 Seventh International Conference on Software Process and Product Measurement*, IEEE, 2012, pp. 99–105.

[132] A.-L. Mesquida, J. Jovanović, M. Jovanović, and A. Mas, "Agile software process improvement: A collaborative game toolbox," *IET Software*, vol. 13, no. 2, pp. 106–111, 2019.

[133] I. van de Weerd and S. Brinkkemper, "Meta-modeling for situational analysis and design methods," in *Handbook of research on modern systems analysis and design technologies and applications*, IGI Global, 2009, pp. 35–54.

[134] P. Kua, *The retrospective handbook*. Leanpub, Victoria, Canada, 2013.

[135] B. W. Tuckman, "Developmental sequence in small groups.," *Psychological bulletin*, vol. 63, no. 6, p. 384, 1965.

[136] S. Beecham, P. O'Leary, S. Baker, I. Richardson, and J. Noll, "Making software engineering research relevant," *Computer*, vol. 47, no. 4, pp. 80–83, 2014.

[137] M. Unterkalmsteiner, T. Gorschek, A. M. Islam, C. K. Cheng, R. B. Permadi, and R. Feldt, "Evaluation and measurement of software process improvement—a systematic literature review," *IEEE Transactions on Software Engineering*, vol. 38, no. 2, pp. 398–424, 2011.

[138] E. M. Rogers, A. Singhal, and M. M. Quinlan, "Diffusion of innovations," in *An integrated approach to communication theory and research*, Routledge, 2014, pp. 432–448.

[139] F. D. Davis, "Perceived usefulness, perceived ease of use, and user acceptance of information technology," *MIS quarterly*, pp. 319–340, 1989.

[140] B. C. Hardgrave, F. D. Davis, and C. K. Riemenschneider, "Investigating determinants of software developers' intentions to follow methodologies," *Journal of management information systems*, vol. 20, no. 1, pp. 123–151, 2003.

[141] J. Iivari, "Why are case tools not used?" *Communications of the ACM*, vol. 39, no. 10, pp. 94–103, 1996.

Appendix A

Invitation to participate

This appendix presents an invitation for participants to present in this research.

Invitation to participate in a research project on requirements documentation in large-scale agile software development projects.

We invite you to participate in a research project that aims to explore how documentation is shaped by requirements engineering activities in multi-team software development projects that adopt agile methods like Scrum, Kanban, SAFe, Scrum@Scale, LeSS, etc.

The objectives of the study are:

- To explore the variations in how large software teams adopt agile methods across different organisations.
- To study how requirements engineering activities are conducted in scaled agile software development approaches. Insights into the RE activities allows us to analyse the current state of documentation artefacts within the project.
- To propose a framework and recommendations to address the identified documentation challenges and to improve documentation practices.

This study is conducted by the student researcher Agnes Wadee, as part of the master thesis for MSc. Business Informatics at Utrecht University, and supervised by Dr. Fabiano Dalpiaz.

What does participation in the study involve?

Participation in the study involves one 45 to 60-minute interview per participant with the researcher. The interview may be conducted online or offline. In addition, documentation analysis on sample documentation artefacts with non-sensitive data will be conducted to study the requirements engineering documentation in the project. We ensure strict confidentiality standards while analysing the documentation.

What are the criteria for participation?

Participating projects are required to meet the following criteria:

1. A multi-team software project that consists of at least two teams, each team having at least five members.
2. The team should be using an agile software development method. This could be pure agile methods such as Scrum, Kanban, XP, etc. Scaled-agile methods such as SAFe, LeSS, Disciplined Agile Delivery, etc. Or an internally created method using a combination of agile activities with other traditional project management methods.
3. The project should concern the development of a software product or service.
4. The commitment to participate in at least 2 interviews per team with senior team members (e.g., scrum master, lead developer, lead architect, lead tester), and one interview with the project manager.
5. The willingness to share sample documentation artefacts with non-sensitive data with the assurance that this information will be handled confidentially and anonymised before analysis and writing of findings.

What is in it for you/your organisation?

Your organisation and the members of the participating project(s) will enjoy the benefits of obtaining insights into their agile development approach, requirements engineering activities and the impact of RE activities on documentation. Not only will these insights be based on the collected data from your organisation, but also the anonymised shared data from the analysis of other participating projects. For example, best practices and lessons learnt can be shared from these insights as a learning

opportunity for your organisation. Also, the challenges of documentation in large scale agile software projects will be analysed for giving recommendations. A framework will be designed to improve the documentation practises in this context and may help improve the state of documentation artefacts and documentation practices in your projects.

Additional information

Participation in the study is voluntary, and the participants can withdraw from the study at any time without consequences. If you agree to participate, you will receive a consent form to be signed. Data confidentiality is guaranteed by keeping the data within the research team, and by using anonymization and pseudonymisation techniques. For more details, see Consent Form.pdf

Interested to participate?

If you are interested in participating in the research, or should you have any questions, please reach out to us via e-mail.

Looking forward to hearing from you!

Kind regards,

Agnes Wadee
a.a.wadee@students.uu.nl

Dr. Fabiano Dalpiaz
f.dalpiaz@uu.nl

Appendix B

Consent form

In this appendix, the consent form is presented.

Informed consent form

The research project:

Title: **Exploring Requirements Engineering Practices in Agile Software Development in Large Software Organisations with focus on documentation**

Researchers and responsible institution:
Agnes Wadee (Thesis researcher, Utrecht University),
Dr. Fabiano Dalpiaz (First Supervisor, Utrecht University)

Project Description:
This research is conducted as part of the requirements of the master's thesis in Business Informatics at the Department of Information and Computing Sciences, Utrecht University. The main research question of the study is phrased as*: "How is documentation impacted by Requirements Engineering practices in Agile Software Development in Large Software Organisations?".*

The purpose of the research is threefold:

1. We aim to gather empirical evidence of the variation of agile development methods in large software projects, i.e., how agile is adopted differently in large-scale software projects. Data concerning the employed requirements engineering (RE) practices are collected to study the use of RE in large scale agile software development as well as the impact of RE activities on documentation.
2. Insights into the RE activities allows us to study the current state of documentation artefacts within the project. Documentation in the context of this research includes both formal documentation such as specification documents, but also semi-formal and/or informal documentation such as the product backlog, sprint backlog, user stories, bugs descriptions, etc. With these, we seek to analyse the challenges facing RE-related documentation in large scale agile software development.
3. We propose a framework and recommendations to address these challenges and improve documentation practices.

Contact person (name, email, phone number):
Agnes Wadee (a.a.wadee@students.uu.nl), Dr Fabiano Dalpiaz (f.dalpiaz@uu.nl)

Participation in the project includes:

- At least 2 interviews per team with senior team members (e.g., scrum master, lead developer, lead architect, lead tester), and one interview with the project manager.
- Documentation analysis of sample documentation artefacts.

Voluntary participation:
Participation is entirely voluntary, and the participant, team, and/or project may withdraw from the study at any time without any negative consequences. In that case, we will inform you about what will happen with the collected data up to the point of withdrawal.

Data confidentiality
Measures will be taken to ensure the confidentiality and anonymity of any data collected in this study. The transcripts and recorded interviews will be kept securely and not be released to any third party other than the first supervisor and researcher. Information on the participation and non-participation of any participating party will be kept confidentially. Pseudonyms will be used for

names of participants, names of projects and/or teams, stakeholders, name and company of clients, and any unique identifier of the software product such as the name of the product. If an internally created method is used, the name of the method, if unique to the company may be pseudo-coded upon request. Your organisation and/or the participating projects have the right to review the information gathered from their project to be published in the thesis for the purpose of data protection compliances.

Further use of the data

I consent to having my data anonymised for use by the aforementioned research team at the Department of Information and Computing Sciences, Utrecht University, for the purpose of this research only.

I hereby confirm with my signature that my questions have been satisfactorily answered by the contact person and that I have read and understood the terms of this consent and participate voluntarily in this project.

Participant

Name, first name:

Place, date, signature:

Contact person

Name, first name:

Place, date, signature:

Appendix C

Questions for semi-structured interview - Type 1

The following questions are used as a guide for the first problem orientation interviews with the project manager.

Prior to the interview

- Ensure consent form has been signed.
- Ask what development method is used in the project for preparation. If it is a standard method, prepare a picture of this method for discussion.
- If available, have documentation on SD method available for the meeting.

Introduction

- Build rapport with the participant.
- Introduce the research project.
- Introduce the interview: in this meeting, we will be discussing your project, the software development methodology as well as how documentation is used within this project.
- Ask for permission to record the meeting.

Project Information

- Can you give me an overview of your project?
- What is your role in the project?
- Can you briefly explain your main responsibilities as a ¡project manager¿ in the project?
- What is the duration of the project?
- In which stage of the SDLC is the project?

Team Information

- How many teams are there in the project?
- How are the teams formed?
- Which roles does each team comprises of?
- How many people are there per team?
- Are the teams located in the same location?
- In which ways do the teams communicate when gathering and clarifying requirements?
- Are there any channels (tools) used to support the communication between teams?
- How do the teams communicate and align the (functionality, infrastructure) with each other?
- Are there activities for the team managers (scrum masters) to align the requirements and progress of the teams (for example, scrum of scrums)?
- How do the teams communicate with the client?

Agile development methods

- How many teams are there in the project?
- What software development methods are you using within the project?
- Can you briefly describe how the (type of method) is used in the project?
- How does (name of project) deviate from the (agile method)?
- Is there documentation on how this method is implemented within (company name) or the project?

Problem orientation

- What kind of documentation and tools are used to elicit (gather) requirements in the project?
- What kind of documentation and tools are used to analyse the requirements in activities such as prototyping, modelling, writing the test plan in the project?
- What kind of documentation and tools are used to document (specify) the agreed requirements in the project?
- How are the agreed requirements validated with the client?
- To what extent do you find that the changes in requirements are reflected in the documentation?
- Who is responsible for writing and maintaining these documentation?
- Are there standardised templates for the documentation? Is the type of documentation standardised across the project teams, or do they differ per team?

- Which documentation is shared across teams? (Both formal and informal artefacts)
- Which kind of documentation is not standardised by the project and differs per team?

Closing

- Thank the participant for his/her time and contribution.
- Get information on who to interview next within the team or project.
- Any open questions.

Appendix D

Questions for semi-structured interview - Type 2

Interview questions type 2

Prior to the interview

- Ensure the consent form has been signed.
- Send the process model for review and preparation

Introduction

- Build rapport with the participant (how are you doing?, etc.)
- Introduce the research project
- Introduce the interview: in this meeting, we will be discussing how documentation is used within this project and the state of these documentation. (Discussions do not have any implication on your job at <company name>, the purpose of this meeting is solely for the purpose of the research)
- Permission to record the meeting: To begin, I would like to ask if you do not object to recording this meeting.

Participant Information

- Can you briefly introduce yourself and your role in this project?

Requirements engineering practices

- For the first part of this interview, we will discuss the activities in the process model and how your work relates to it. [check for missing activity, artefacts, flow, etc.]
    - [If the rationale of use is not yet known] Now that we have understood the process flow, why is <documentation> used in the project?
- What are the main documentation relevant to your activities? (both formal and informal artefacts) (for the context of this interview)
    - Which documentation do you use during those activities? [naming specific activities]
    - Which documentation do you contribute to during those activities?
    - Which documentation are you responsible for?
- Other open questions from the previous interview

Documentation artefacts

Information content (What): captures the issues of what is written in the documentation[2].
Correctness, completeness, up-to-dateness

Correctness ensures that the information provided in the documentation is precise and in accordance with the facts [98].

- How do you ensure that the user stories are written precisely?
- How do you ensure that the user stories adhere to the given template?
- How do you ensure that the *requirements* documents are written precisely?
- How do you ensure that the *requirements* documents adhere to the given template?
    - [Is there any difference in] how you ensure that the *design* documentation is written precisely?
    - [Is there any difference in] how you ensure that the *design* documentation adheres to the given template?
    - [Is there any difference in] how you ensure that the test cases are written precisely? [Is there any difference in] How you ensure that the *test* cases adhere to the given template?
- In your opinion, which documentation is the most problematic to keep correct? (And why?)

Completeness ensures that the documentation provides the information needed by stakeholders about the system or its modules to perform their tasks [98].

- Are all the requirements in the documentation, or are there other (unwritten) sources? (Is the <documentation> complete without any missing information?)
    - Is that the same situation for the design documentation?
    - Is that the same situation for the test documentation?

- How accurate and complete are the references to other documentation in the user story in Azure DevOps?

Up-to-dateness ensures that the documentation is in sync with the other parts of the system [2]. A major difference with the other two criteria is the information can be correct and complete prior to the introduction of a change [2].

- Is the documentation consistent with the working software/ product increment?
- Do you know of any functionality in the code that is not yet documented?
- Do you think that the documentation is sufficient to extend or maintain the application?
- (Only if they have translations) Are there any translations in the documentation that are outdated?
- What are some of the challenges faced when ensuring documentation is kept up-to-date?
- Is it easy to trace the versioning and updates of the documentation?

Information content (How) discusses problems relating to the writing style and organisation of documentation [2].

Maintainability, readability, usability, usefulness

Maintainability refers to the degree of ease to apply changes to documentation [2].

- Can the requirements be found in many sources or is there only one source of truth? (duplicates)
    - Is this situation the same for the test cases?
    - Is this situation the same for the design specifications/architecture documentation?
- How easy is it to add changes to the requirements documentation (including the user stories)? Do you also have to update other documentation?
    - Is this situation the same for the test cases?
    - Is this situation the same for the design specifications/architecture documentation?
- Is there a process in place to know how a change in one documentation such as the (requirements/test cases/ design specification, architecture) impacts the others?
- [if not discussed] What are the challenges of incorporating change requests in the documentation?

Readability defines the degree of ease to read a document.

- Are there processes in place to review documentation and ensure that they are clear to read?
- In your opinion, how concise are the contents of the documentation?

Usability "Usability of documentation refers to the degree to which it can be used by readers to achieve their objectives effectively" [2, p. 1205].

- Are there any issues faced when using Azure DevOps/other tools such as drawing programs to write documentation? If so, what are the workarounds
- Is it difficult for the project members and the client to find information in the various documentation?
- Do writers adhere to the best practices and templates for writing the documentation?
- Are there processes in place to ensure shared documentation is maintained correctly by both teams? (If any shared documentation)

Usefulness defines whether the documentation "is of practical use to its readers" [2, p. 1205]. Feedback from various stakeholders is therefore crucial to ensure useful documentation [2].

- Are there processes in place to encourage the various stakeholders to review and give feedback on the documentation?
- How is feedback planned and incorporated into the documentation process?

Process-related discusses issues relating to the documentation process. The sub-categories are internationalisation, contribution to documentation, doc-generator configuration, development issues caused by documentation, and traceability.

- (If there is a need for translation) what are the difficulties faced in translating and reviewing the translation of the documents?
- Are there processes in place to ensure the planning of documentation to ensure it is not forgotten due to lack of time?
- Is there a process in place to report issues found in documentation? (Are there issues found in the documentation assigned a priority?)Is there support in place for external contributors to the documentation (such as clients)?
- Is any of the documentation automatically generated?
- What are the challenges facing the process of writing documentation?
- What is going well with regards to the documentation process in your opinion?
-

Closing

- Thank the participant for his/her time and contribution
- Any open questions?

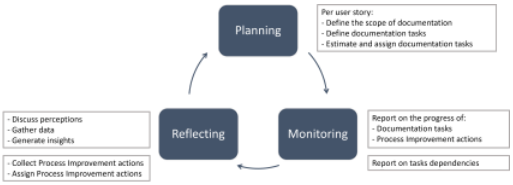
Appendix E

Consent form of experimental study

In this appendix section, we present the consent form for the participants of the experiment.

Introduction

Dear participant,
Thank you for participating in this research thus far!
The purpose of this survey is to gather your perceptions of the application of the DIFFC model within your team during the past sprint.

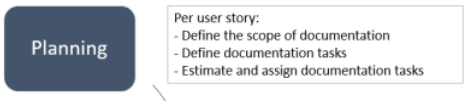


This is a short survey and will take approximately 6-8 minutes of your time. Please answer all the required questions. You may press 'Next' to continue.

Kind regards,
Agnes Wadee

Planning phase

These questions relate to your perceptions of the effectiveness of the use **Planning Phase** of the DIFFC model during the Sprint Planning of Sprint 27.



Ease of use

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
I found it easy to apply the Planning Phase of the DIFFC model to the Sprint Planning.	○	○	○	○	○

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The guidelines provided by the Planning Phase were clear and understandable.	○	○	○	○	○
I found it easy to adapt the guidelines of the Planning Phase to the Sprint Planning.	○	○	○	○	○
It was easy for me to become skilful at using the guidelines of the Planning Phase.	○	○	○	○	○

Productivity

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The use of the Planning phase has sped up the process of updating documentation within the past sprint.	○	○	○	○	○
The use of the Planning phase has made documentation more measurable within the past sprint.	○	○	○	○	○
The use of the Planning Phase has increased my productivity with regard to keeping documentation up-to-date.	○	○	○	○	○

Quality

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The use of the Planning Phase has enhanced the quality of documentation during the past sprint.	○	○	○	○	○
The use of the Planning Phase has minimised the issue of forgotten documentation during the past sprint.	○	○	○	○	○
The use of the Planning Phase has clarified exactly what needs to be documented per user story.	○	○	○	○	○
The use of the Planning Phase has improved the overall quality of the Sprint Planning.	○	○	○	○	○
The use of the Planning Phase has made me more aware of documentation quality.	○	○	○	○	○

Usefulness

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
Using the Planning Phase improved the overall state of documentation within the past sprint.	○	○	○	○	○
Using the Planning Phase added more structure to the way we approach documentation as a team.	○	○	○	○	○
The Planning phase integrated well into our way of working as a team.	○	○	○	○	○
I found the guidelines prescribed by the Planning Phase useful to my job.	○	○	○	○	○

Use

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
I would keep using the activities of the Planning Phase within my team.	○	○	○	○	○
I would recommend the guidelines of the Planning Phase to other teams within the company.	○	○	○	○	○

Any other remarks on the Planning Phase:

Monitoring phase

These questions relate to your perceptions of the effectiveness of the use **Monitoring Phase** of the DIFFC model during the Daily Stand-ups of Sprint 27.

Ease of use

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
I found it easy to apply the Monitoring Phase of the DIFFC model to the Daily Stand-up meetings.	○	○	○	○	○

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The guidelines provided by the Monitoring Phase were clear and understandable.	○	○	○	○	○
I found it easy to adapt the guidelines of the Monitoring Phase to the Daily Stand-up meetings.	○	○	○	○	○
It was easy for me to become skilful at using the guidelines of the Monitoring Phase.	○	○	○	○	○

Productivity

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The use of the Monitoring phase has sped up the process of updating documentation within the past sprint.	○	○	○	○	○
The use of the Monitoring phase has made documentation more measurable within the past sprint.	○	○	○	○	○
The use of the Monitoring Phase has increased my productivity with regard to keeping documentation up-to-date.	○	○	○	○	○

Quality

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The use of the Monitoring Phase has enhanced the quality of documentation during the past sprint.	○	○	○	○	○
The use of the Monitoring Phase has minimised the issue of forgotten documentation during the past sprint.	○	○	○	○	○
The use of the Monitoring Phase has improved the overall quality of the Daily Stand-up meetings.	○	○	○	○	○
The use of the Monitoring Phase has made me more aware of documentation quality.	○	○	○	○	○

Usefulness

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
Using the Monitoring Phase improved the overall state of documentation within the past sprint.	○	○	○	○	○

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
Using the Monitoring Phase added more structure to the way we approach documentation as a team.	○	○	○	○	○
The Monitoring Phase integrated well into our way of working as a team.	○	○	○	○	○
I found the guidelines prescribed by the Monitoring Phase useful to my job.	○	○	○	○	○

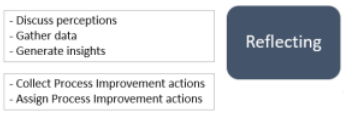
Use

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
I would keep using the activities of the Monitoring Phase within my team.	○	○	○	○	○
I would recommend the guidelines of the Monitoring Phase to other teams within the company.	○	○	○	○	○

Any other remarks on the Monitoring Phase:

Reflecting phase

These questions relate to your perceptions of the effectiveness of the use **Reflecting Phase** of the DIFFC model during the Retrospective of Sprint 26 and 27.



Ease of use

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
I found it easy to apply the Reflecting Phase of the DIFFC model to the Retrospective sessions.	○	○	○	○	○

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The guidelines provided by the Reflecting Phase were clear and understandable.	○	○	○	○	○
I found it easy to adapt the guidelines of the Reflecting Phase to the Retrospective sessions.	○	○	○	○	○
It was easy for me to become skilful at using the guidelines of the Reflecting Phase.	○	○	○	○	○

Productivity

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The use of the Reflecting phase has sped up the process of updating documentation within the past sprint.	○	○	○	○	○
The use of the Reflecting phase has made documentation more measurable within the past sprint.	○	○	○	○	○
The use of the Reflecting Phase has increased my productivity with regard to keeping documentation up-to-date.	○	○	○	○	○

Quality

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The use of the Reflecting Phase has enhanced the quality of documentation during the past sprint.	○	○	○	○	○
The use of the Reflecting Phase has minimised the issue of forgotten documentation during the past sprint.	○	○	○	○	○
The use of the Reflecting Phase has improved the overall quality of the Retrospective sessions.	○	○	○	○	○
The use of the Reflecting Phase has made me more aware of documentation quality.	○	○	○	○	○

Usefulness

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
Using the Reflecting Phase improved the overall state of documentation within the past sprint.	○	○	○	○	○

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
Using the Reflecting Phase added more structure to the way we approach documentation as a team.	○	○	○	○	○
The Reflecting Phase integrated well into our way of working as a team.	○	○	○	○	○
I found the guidelines prescribed by the Reflecting Phase useful to my job.	○	○	○	○	○

Use

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
I would keep using the activities of the Reflecting Phase within my team.	○	○	○	○	○
I would recommend the guidelines of the Reflecting Phase to other teams within the company.	○	○	○	○	○

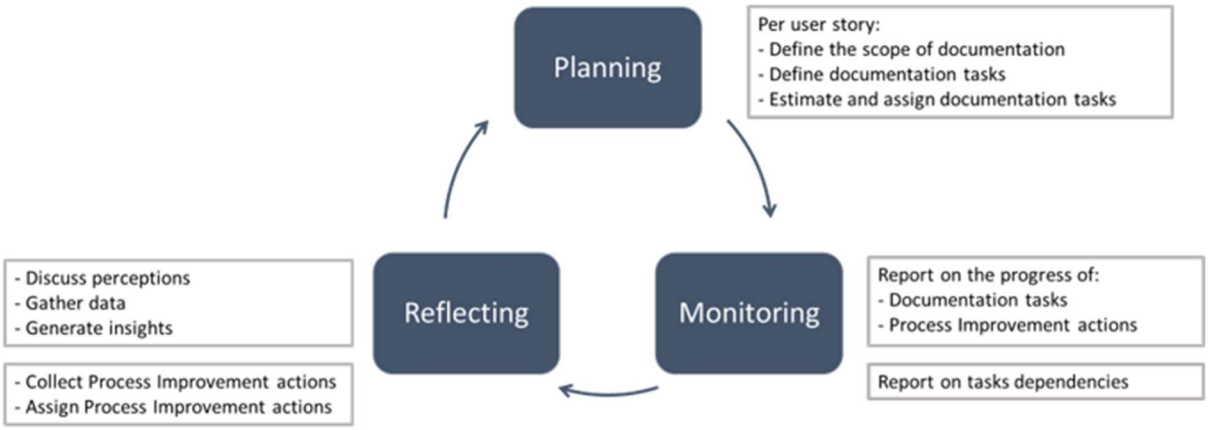
Any other remarks on the Reflecting Phase:

Appendix F

Emailed guidelines for experiment

In this appendix section, we present an overview of the email with guidelines that was sent to the Scrum Masters and/or Scrum Master representative of each team.

Dear [Scrum Master],
Thank you once again for your wiliness to help out in the validation research. Here is some additional information about the observations from next week.



2. In the Sprint Planning, the team will incorporate the following into the existing process.
   For each user story:
   a. Define the scope of the documentation
   b. Define documentation task(s)
   c. Estimate and assign documentation tasks (responsible, and reviewer)

   I will be observing the Sprint Planning scheduled for next week
3. In the Daily-Stand-up, as progress on user stories and tasks are already been reported upon, the team will incorporate the reporting of progress on:
   a. Documentation tasks
   b. Tasks dependencies hindering the progress of documentation task
   c. Process improvement actions (from the retrospective)
4. In the Retrospectives, the team will incorporate the following:
   a. Discussing perceptions of the previous sprint (this is already being done)
   b. Gathering data. For each of your teams, would you be able to decide on a subject for discussion?, for example 'Reducing documentation debt',
      Then, gather some data on the documentation debt on the backlog?
   c. Afterwards, you can visualise the data for the retrospective.
   d. Then, during the retrospective, we will use the game-based approach to stimulate idea generation.

Lastly, if an activity from these phases already exists, it does not need to be introduced again.

I will be observing the meetings and taking notes (and not recording, so the team may speak freely 😊), so I will be available for questions.

In this link, a consent form is provided for all participants, please have a look at it and I will appreciate if you can forward it to your team, so they can fill it in before the first observation.
Here is a link to the consent form https://survey.uu.nl/jfe/form/SV_3xagw6Us324slwO

Kind regards,
Agnes Wadee

Appendix G

Prepared Retrospective Board using the Role-Expectation Matrix game

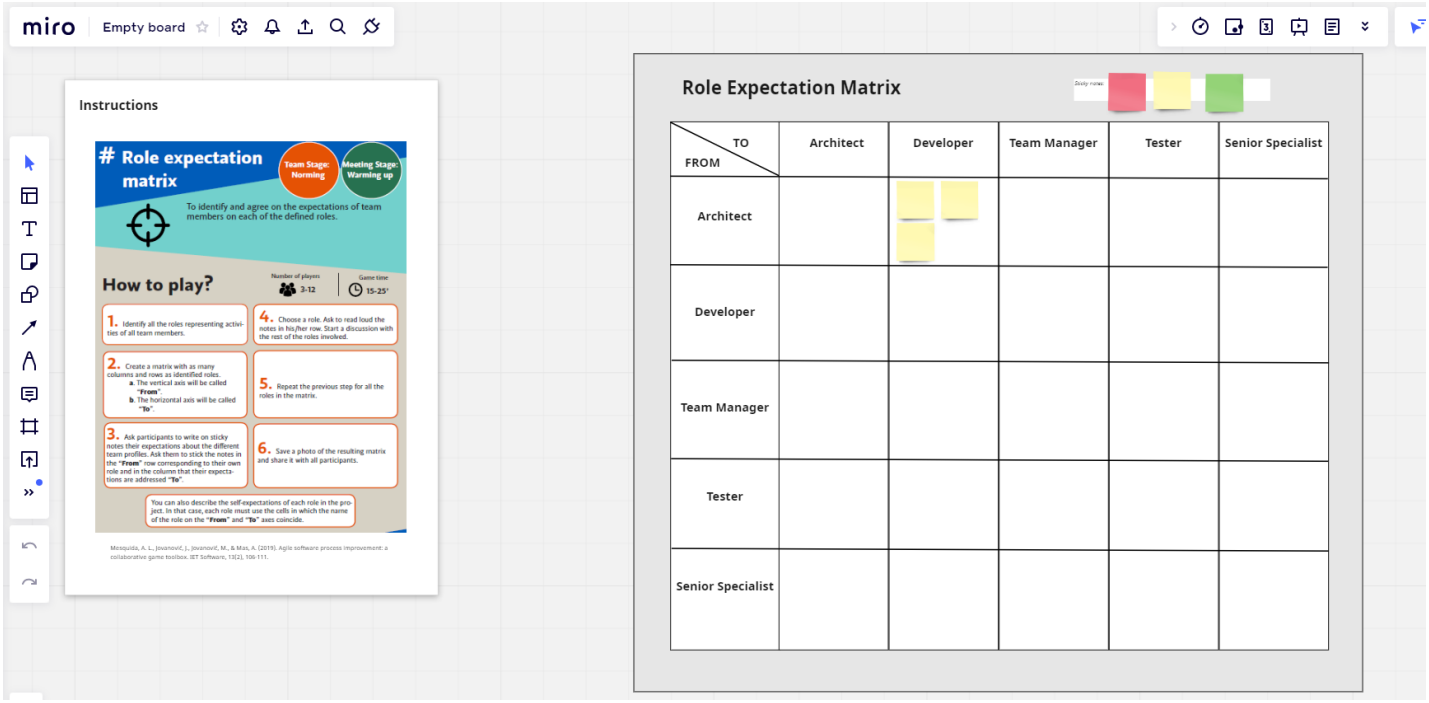


Figure G.1: A screenshot of the Retrospective board for sprint n.

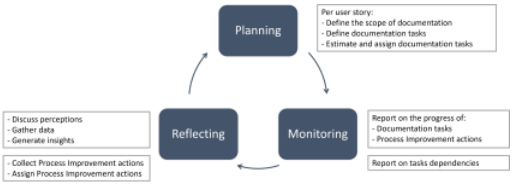
Appendix H

Validation Survey

In this appendix section, we present the validation survey for the participants.

Introduction

Dear participant,
Thank you for participating in this research thus far!
The purpose of this survey is to gather your perceptions of the application of the DIFFC model within your team during the past sprint.

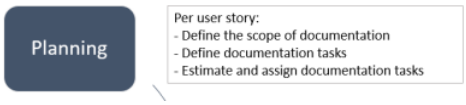


This is a short survey and will take approximately 6-8 minutes of your time. Please answer all the required questions. You may press 'Next' to continue.

Kind regards,
Agnes Wadee

Planning phase

These questions relate to your perceptions of the effectiveness of the use **Planning Phase** of the DIFFC model during the Sprint Planning of Sprint 27.



Ease of use

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
I found it easy to apply the Planning Phase of the DIFFC model to the Sprint Planning.	○	○	○	○	○

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The guidelines provided by the Planning Phase were clear and understandable.	○	○	○	○	○
I found it easy to adapt the guidelines of the Planning Phase to the Sprint Planning.	○	○	○	○	○
It was easy for me to become skilful at using the guidelines of the Planning Phase.	○	○	○	○	○

Productivity

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The use of the Planning phase has sped up the process of updating documentation within the past sprint.	○	○	○	○	○
The use of the Planning phase has made documentation more measurable within the past sprint.	○	○	○	○	○
The use of the Planning Phase has increased my productivity with regard to keeping documentation up-to-date.	○	○	○	○	○

Quality

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The use of the Planning Phase has enhanced the quality of documentation during the past sprint.	○	○	○	○	○
The use of the Planning Phase has minimised the issue of forgotten documentation during the past sprint.	○	○	○	○	○
The use of the Planning Phase has clarified exactly what needs to be documented per user story.	○	○	○	○	○
The use of the Planning Phase has improved the overall quality of the Sprint Planning.	○	○	○	○	○
The use of the Planning Phase has made me more aware of documentation quality.	○	○	○	○	○

Usefulness

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
Using the Planning Phase improved the overall state of documentation within the past sprint.	○	○	○	○	○
Using the Planning Phase added more structure to the way we approach documentation as a team.	○	○	○	○	○
The Planning phase integrated well into our way of working as a team.	○	○	○	○	○
I found the guidelines prescribed by the Planning Phase useful to my job.	○	○	○	○	○

Use

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
I would keep using the activities of the Planning Phase within my team.	○	○	○	○	○
I would recommend the guidelines of the Planning Phase to other teams within the company.	○	○	○	○	○

Any other remarks on the Planning Phase:

Monitoring phase

These questions relate to your perceptions of the effectiveness of the use **Monitoring Phase** of the DIFFC model during the Daily Stand-ups of Sprint 27.

Ease of use

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
I found it easy to apply the Monitoring Phase of the DIFFC model to the Daily Stand-up meetings.	○	○	○	○	○

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The guidelines provided by the Monitoring Phase were clear and understandable.	○	○	○	○	○
I found it easy to adapt the guidelines of the Monitoring Phase to the Daily Stand-up meetings.	○	○	○	○	○
It was easy for me to become skilful at using the guidelines of the Monitoring Phase.	○	○	○	○	○

Productivity

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The use of the Monitoring phase has sped up the process of updating documentation within the past sprint.	○	○	○	○	○
The use of the Monitoring phase has made documentation more measurable within the past sprint.	○	○	○	○	○
The use of the Monitoring Phase has increased my productivity with regard to keeping documentation up-to-date.	○	○	○	○	○

Quality

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The use of the Monitoring Phase has enhanced the quality of documentation during the past sprint.	○	○	○	○	○
The use of the Monitoring Phase has minimised the issue of forgotten documentation during the past sprint.	○	○	○	○	○
The use of the Monitoring Phase has improved the overall quality of the Daily Stand-up meetings.	○	○	○	○	○
The use of the Monitoring Phase has made me more aware of documentation quality.	○	○	○	○	○

Usefulness

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
Using the Monitoring Phase improved the overall state of documentation within the past sprint.	○	○	○	○	○

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
Using the Monitoring Phase added more structure to the way we approach documentation as a team.	○	○	○	○	○
The Monitoring Phase integrated well into our way of working as a team.	○	○	○	○	○
I found the guidelines prescribed by the Monitoring Phase useful to my job.	○	○	○	○	○

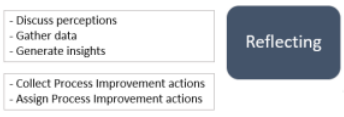
Use

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
I would keep using the activities of the Monitoring Phase within my team.	○	○	○	○	○
I would recommend the guidelines of the Monitoring Phase to other teams within the company.	○	○	○	○	○

Any other remarks on the Monitoring Phase:

Reflecting phase

These questions relate to your perceptions of the effectiveness of the use **Reflecting Phase** of the DIFFC model during the Retrospective of Sprint 26 and 27.



Ease of use

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
I found it easy to apply the Reflecting Phase of the DIFFC model to the Retrospective sessions.	○	○	○	○	○

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The guidelines provided by the Reflecting Phase were clear and understandable.	○	○	○	○	○
I found it easy to adapt the guidelines of the Reflecting Phase to the Retrospective sessions.	○	○	○	○	○
It was easy for me to become skilful at using the guidelines of the Reflecting Phase.	○	○	○	○	○

Productivity

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The use of the Reflecting phase has sped up the process of updating documentation within the past sprint.	○	○	○	○	○
The use of the Reflecting phase has made documentation more measurable within the past sprint.	○	○	○	○	○
The use of the Reflecting Phase has increased my productivity with regard to keeping documentation up-to-date.	○	○	○	○	○

Quality

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The use of the Reflecting Phase has enhanced the quality of documentation during the past sprint.	○	○	○	○	○
The use of the Reflecting Phase has minimised the issue of forgotten documentation during the past sprint.	○	○	○	○	○
The use of the Reflecting Phase has improved the overall quality of the Retrospective sessions.	○	○	○	○	○
The use of the Reflecting Phase has made me more aware of documentation quality.	○	○	○	○	○

Usefulness

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
Using the Reflecting Phase improved the overall state of documentation within the past sprint.	○	○	○	○	○

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
Using the Reflecting Phase added more structure to the way we approach documentation as a team.	○	○	○	○	○
The Reflecting Phase integrated well into our way of working as a team.	○	○	○	○	○
I found the guidelines prescribed by the Reflecting Phase useful to my job.	○	○	○	○	○

Use

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
I would keep using the activities of the Reflecting Phase within my team.	○	○	○	○	○
I would recommend the guidelines of the Reflecting Phase to other teams within the company.	○	○	○	○	○

Any other remarks on the Reflecting Phase: