Lappeenranta-Lahti University of Technology LUT
School of Engineering Science
Software Engineering
Master's Programme in Software Engineering and Digital Transformation

Anu Vanhanen

DOCUMENTATION OF THE PRODUCT DURING A SOFTWARE DEVELOPMENT PROCESS - THE SCALED AGILE FRAMEWORK

Examiners: Professor Jari Porras
D. Sc. (Tech) Ari Happonen

Supervisors: Riikka Mänttäri
Professor Marko Torkkeli

ABSTRACT

Lappeenranta-Lahti University of Technology LUT
School of Engineering Science
Software Engineering
Master's Programme in Software Engineering and Digital Transformation

Anu Vanhanen

Documentation of the Product During a Software Development Process – The Scaled Agile Framework

Master's Thesis
2019

90 pages, 14 figures, 3 tables, 2 appendixes

Examiners: Professor Jari Porras
D. Sc. (Tech) Ari Happonen

Keywords: Agile software development, agile, documentation, product documentation

Documentation is an essential part of software development processes, although as the use of agile methods has increased the role of documentation has become unclear, because agile practitioners argue against comprehensive documentation. The issue of how much documentation is enough and in which development phase should it be created has been studied for years, yet it still remains unclear. This thesis aims to find guidelines of product documentation when the company is using agile methods. The focus of the thesis is on case company's product documentation, and the goal is to offer a possible future solution related to the documentation of their product. In the empirical study it was found out that requirements, functionalities, features, configurations and interfaces are important things to document. Wiki and UML diagrams were noted to be good documentation techniques.

TIIVISTELMÄ

Lappeenrannan-Lahden teknillinen yliopisto
School of Engineering Science
Tietotekniikan koulutusohjelma
Master's Programme in Software Engineering and Digital Transformation

Anu Vanhanen

Tuotteen dokumentointi ohjelmistotuotantoprosessin aikana – Viitekehyksenä SAFe

Diplomityö
2019

90 sivua, 14 kuvaa, 3 taulukkoa, 2 liitettä

Työn tarkastajat: Professori Jari Porras
Tutkijatohtori Ari Happonen

Hakusanat: Ketterä ohjelmistokehitys, ketteryys, dokumentaatio, tuotteen dokumentointi
Keywords: Agile software development, agile, documentation, product documentation

Dokumentaatio on tärkeä osa ohjelmistotuotantoa, mutta ketterien menetelmien käytön yleistyminen on tuonut epävarmuutta dokumentoinnin rooliin, sillä ketterien menetelmien harjoittajat ovat sitä mieltä, ettei kattavaa dokumentaatiota tarvita. Sitä kuinka paljon dokumentaatiota tulisi tuottaa ja missä prosessin vaiheessa on tutkittu monia vuosia, mutta yksiselitteistä vastausta ei ole löydetty. Tämä diplomityö pyrkii löytämään suuntaviivoja ketterälle tuotedokumentaatiolle. Työ keskittyy case-yrityksen tuotteen dokumentointiin, ja työn tavoitteena on antaa ehdotus siitä, miten dokumentaatioprosessia voitaisiin yrityksessä lähteä kehittämään. Empiirisen tutkimuksen perusteella saatiin selville, että vaatimusten, toiminnallisuuksien, ominaisuuksien, konfiguraatioiden ja rajapintojen dokumentointi on tärkeää. Lisäksi Wiki ja UML-diagrammit todettiin hyviksi dokumentointitekniikoiksi.

ACKNOWLEDGEMENTS

First, I would like to thank my supervisor from the case company, Riikka Mänttäri, for guiding me during this thesis project, but also for challenging me and "forcing" me to do my best. I would also like to thank everyone from the case company who participated in the interviews. In addition to the case company, I would like to thank the examiners of this thesis, Ari Happonen and Jari Porras, for their help and guidance.

Last but not least, I would like to thank my family and friends for offering me support and encouraging me during this whole project. It truly meant a lot.

TABLE OF CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

AM	Agile Modeling
ER	Entity-Relationship
IAM	Identity and Access Management
IDE	Integrated Development Environment
QA	Quality Assurance
RE	Requirements Engineering
ROI	Return of Investment
SAFe	The Scaled Agile Framework
SDLC	Software Development Life Cycle
TCO	Total Cost of Ownership
UI	User Interface
UML	Unified Modeling Language
VCS	Version Control System
XP	Extreme Programming

1 INTRODUCTION

In many organizations software documentation is practiced incompetently (Garousi et al., 2013). Documentation tends to be out-of-date, inaccurate, lacking information or it just simply doesn't exist (Steinberger and Prakash, 2011). The amount of documentation produced in the companies varies widely and documents tend not to be understandable. When companies use agile methodologies, which are people-oriented methods where communication plays a vital role, it makes it even more complicated considering documentation. It is widely argued whether agile methods are producing too little documentation or just enough (Wagenaar et al., 2018; Rico et al., 2009).

Documentation is an integral distribution in order to create and maintain software systems, however there are some limitations related to it. It's expensive to create and maintain documentation, and at the same time documentation is a non-executable artifact, which existence won't directly affect the development of a software product. (Robillard et al., 2017) Agile methods encourage the face-to-face communication rather than documentation, but as Stettina and Heijstek (2011) point out, verbal communication is vulnerable to mistakes of memory and after some time it gets harder to remember the reasons behind the decisions, if they only exist in people's minds. There are also limitations related to the human memory such as duration, capacity to store information and the reliability of the memory (Hakkarainen et al., 2014), which alone could be seen as qualified reasons for documentation. However, even though the issue of how much documentation should be produced has been studied many years, it's still not clear. Therefore, deciding about the amount and extent of the documentation is still a major challenge for many teams, which on the other hand leads to situations where either too much or too little documentation is produced. (Garousi et al., 2013)

1.1 Background

This thesis concentrates on the documentation of the product of the case company and aims to offer a solution of how the case company could start improving their documentation process. This thesis views documentation from the point of view of agile methods since the case company is currently using the Scaled Agile Framework (SAFe), which is a framework

that combines lean and agile methodologies (Scaled Agile, Inc., 2018). Although this thesis concentrates on the product documentation of the case company's product, the literature section views product documentation during agile software development processes in general. This thesis project started because the case company wants to improve their documentation process and wanted to find out which things might be worth doing differently in the future. The expectation of this thesis was to provide some kind of guidance on how to make the documentation process as easy as possible for everyone yet remain the agility in a way that everyone benefits from the documentation.

Figure 1 shows the main aspects, which are taken into account in this thesis. The main aspects include: Agile as a model of software development process; SAFe as a framework; Scrum, Kanban, XP, and Lean as agile methodologies; and agile modeling in addition to these methods to bring the perspective of agile documentation. The focus of the thesis can be summed up as: documentation of the product in an agile way.

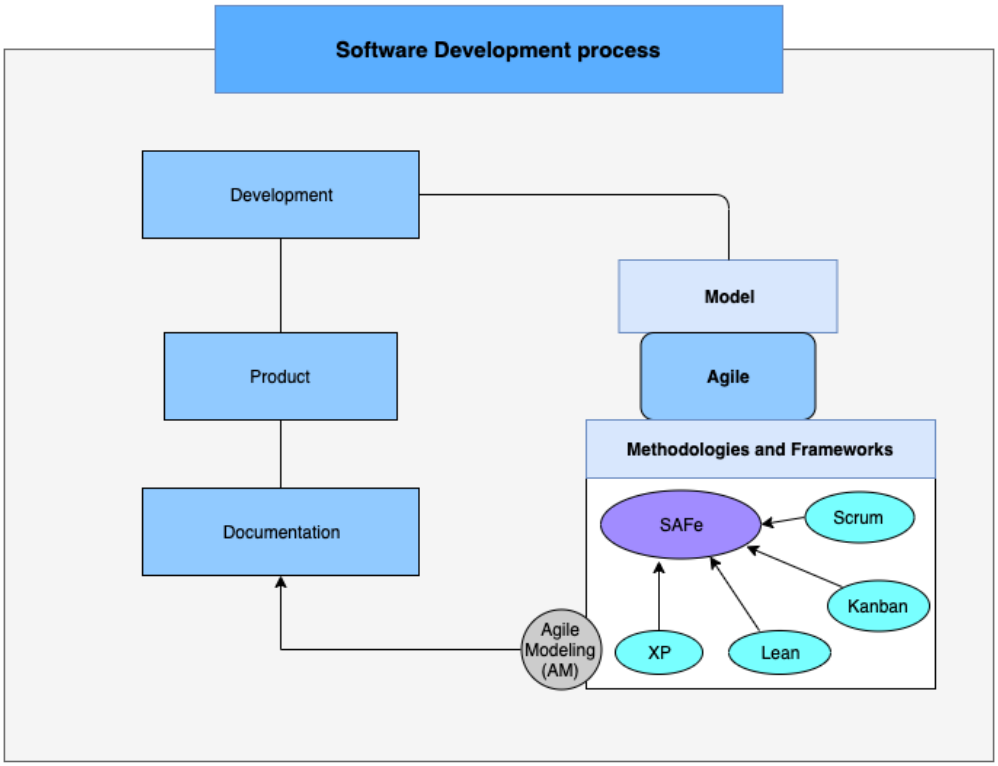

Figure 1. The main aspects that are addressed in the thesis

1.2 Goals and delimitations

The goal of this thesis is to provide a proposition of how the organization could start improving their documentation related to the product. The research is limited to only product documentation and therefore process documentation is not considered. The literature review part is limited to projects where agile methods have been used. The aim of the research is to find answers to the following research questions:

- What should be documented related to the product during its life cycle in agile software development process?
- How and when should the documentation related to the product be created?
    - Which tools should be used?
    - Which style of documenting should be used?
- Who should produce the documentation?

1.3 Research method

The research methodology of this thesis consists of two parts: literature review and empirical study. The literature review part was conducted by snowballing method, and the empirical part was carried out by interviews and survey. Interview was chosen as a method to collect data for empirical part since it was important to get the opinions of the actual stakeholders of the product. Literature research was conducted in order to find out what the researchers have discovered about the topic.

As mentioned before, snowballing was chosen as a research method for the literature review part. Snowballing is an approach where the reference list of a certain paper or the citations to the paper are used to discover and identify new references. Snowballing consists of backward snowballing and forward snowballing. In backward snowballing the reference list of a paper is examined, and in forward snowballing the citations to the paper are explored. (Wohlin, 2014) Google Scholar was chosen as the source for the material search since it doesn't prefer any publishers (Wohlin, 2014), and because its search function is actually

quite good. Time frame of 2001-2019 was chosen, and the reason behind choosing the start year of the time frame lies in the fact that Agile Manifesto was formed in 2001.

At first the start set of papers was defined. The following string of words was used to search material from Google Scholar: *"Documentation in agile software development processes"*, however it was soon noticed that the majority of papers that were found didn't have the word "documentation" in the title. In order to specify the search, the next search string was shortened to: *"Agile documentation"*. These two searches were done in order to define the start set of papers. With the first search nine candidate papers were identified, and five of them were included in the start set. With the second search seven candidate papers were identified, and four of those were included in the start set. The start set of papers was formed from nine papers. The start set of papers looked quite good since the papers were from time frame of 2003-2011, and any author didn't appear in more than one paper.

After defining the start set, backward and forward snowballing was conducted to all of the nine papers in the start set. In backward snowballing phase, the references of the start set papers were reviewed in order to see if there were more papers that could have been included. Overall, the nine papers of the start set have 215 references. From these references 96 were excluded because of the publication year, and 62 references were excluded because they didn't fit the scope of the thesis. After this exclusion there were 57 possible references to be included. After taking a closer look at the 57 references, 15 of them were included. Overall, after the backward snowballing the list of included papers consisted of 24 references.

After backward snowballing, forward snowballing was executed. In forward snowballing phase, the papers that are citing the nine papers of the start set were reviewed in order to see if there were references that could have been included. Because of the chosen time frame of 2001-2019, there were a lot of citations to the start set of papers, which are published between 2003-2011. Over 2500 papers cite the nine papers from the start set, however one of the start set papers is cited over 1300 times and one is cited over 500 times. The title of the paper cited over 1300 is "*Agile Software Development: The Business of Innovation"*, and that is why many of the citing papers were out of the scope of this thesis because of lacking information related to the documentation. Because of the huge amount of citing papers, it

wasn't possible to go through all of them. Most of the papers were judged by the title. Overall 18 papers were included during the forward snowballing.

After both backward and forward snowballing, the reference list included 42 references. It was decided that only one iteration of snowballing would be enough because of already wide range of references. The included references also cover the time frame of 2001-2018. In addition to these references few books, websites and papers were included later. These references were searched separately because of a specified need of certain type of information.

As mentioned before the empirical part of this research was executed by interviewing the employees of the case company. Overall 18 interviewees participated, and 15 interviews were held, one of which was a group interview with four participants. Most of the interviews were executed by face-to-face interviews, but few interviews were held via Skype, because of the location of the interviewee. Each participant also filled a survey form related to the product documentation. The interviews consisted of seven interview questions, which were hoped to help in answering the research questions of the thesis. The interviews were indeed successful and served their purpose.

1.4 Structure of the thesis

This first chapter was the introduction to this thesis project. The background and reason for the thesis project were introduced. Also, the goals and delimitations of this thesis were presented. The used research method was also explained. Chapters two and three are covering the literature review of this research. Chapter two provides a brief overview related to the agile methodologies, and chapter three concentrates on documentation and how the use of agile methods affect the documentation process. Chapter four covers the qualitative research and the conducted empirical study is presented in this chapter. In chapter five the proposed solution about how the case company could start producing the documentation related to their product in the future is introduced and discussed. Chapter six covers the conclusions and summary of the thesis.

2 AGILE METHODOLOGIES

The Purpose of this chapter is to provide a very brief introduction to certain agile methods, models and frameworks since the case company is using the Scaled Agile Framework (SAFe). This thesis focuses on the documentation of a software product during the software development process. Documentation is being viewed from the point of view of agile development processes since the case company is using SAFe, which is a framework combining lean and agile methodologies.

In the subchapters the most important agile methods from the point of view of this thesis are shortly introduced. These include: Scrum, Kanban, Extreme Programming (XP), and Lean. Also, Agile Modeling is introduced since it focuses on documentation and is that way really closely related to the topic of this thesis. SAFe is introduced because it is the framework that the company is using. Subchapter 2.1 introduces some principles from Agile Manifesto from the point of view of documentation, subchapter 2.2 introduces the agile methods (Scrum, Kanban, XP and Lean), subchapter 2.3 concentrates on Agile Modeling, and subchapter 2.4 introduces the Scaled Agile Framework.

2.1 Agile Manifesto - What It Says About Documentation?

The manifesto for agile software development goes as follows (Beck et al., 2001):

”We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

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

That is, while there is value in the items on the right, we value the items on the left more.”

"Working software over comprehensive documentation", even the agile manifesto itself says that documentation is not emphasized in agile software development processes. There is value in comprehensive documentation, but the authors of the Agile Manifesto put more value on working software (Wagenaar et al., 2018). However, two authors of Agile Manifesto, Fowler and Highsmith (2001), also point out that comprehensive documentation is not necessarily bad, but the main focus must still be the delivery of working software. This means that it is up to each project team to determine the documentation that is absolutely necessary to produce (Fowler and Highsmith, 2001).

2.2 Agile Methods

Agile methodologies are designed to support early and quick production of the code. That is why agile development processes are structured into iterations. The purpose of an iteration is to focus on delivering working code and other artifacts that are valuable to the customer, and also to the project. (Turk et al., 2002) Agile methods focus on continuous interaction with the customer, and the aim is to be able to deal with the changing requirements. Agile methods also focus on prioritizing the requirements and delivering the most valuable functionalities first. (Sillitti and Succi, 2005)

2.2.1 Scrum

Scrum is an agile management method that is used in software development projects to manage the product development. (Pfahl, 2014; Agile Alliance, 2019) Scrum's main contribution to the software development is an approach that is simple yet effective for managing the work of a small team involved in the product development process. (Agile Alliance, 2019) Scrum is best fit in the cases where a team is working in a product development environment where the work can be split into more than one 2-4 week iteration. These iterations, short work cycles, which are called sprints are one of the main elements in Scrum processes. The sprint is a timebox during which the team produces a product increment that can potentially be shipped. (Agile Alliance, 2019; Pfahl, 2014) One of the main characteristics of sprints is that the new sprint follows directly after the conclusions of the former sprint. (Agile Alliance, 2019)

2.2.2 Kanban

The Kanban method is an agile methodology which helps to design, manage, and improve the work. The Kanban method gets its name from the use of Kanban board, where the work items are visually presented on, allowing the whole team to see every state of work at any time. In Kanban the work "flows" continuously all the time instead of being organized into timeboxes like in Scrum. (Agile Alliance, 2019; Radigan, 2019) Kanban is a popular method used by agile teams because it offers several advantages for the team such as planning flexibility and shortened time cycles. A Kanban team only focuses on active work that is in progress and once a work item is complete, the team moves into the next work item which is on the top of the backlog. That is why it is important that the most important work items are on top of the backlog. (Radigan, 2019) The main idea of a Kanban board is that it shows the whole team which work tasks are completed, which are in progress and which are waiting on the backlog. The board also shows the important tasks that should be taken care of as soon as possible, and also who is responsible of the tasks.

2.2.3 Extreme Programming (XP)

Extreme Programming (XP) is an agile software development method with an aim to produce high quality softwares (Agile Alliance, 2019). Beck (2000) describes XP as a light-weight methodology for small to medium-sized software development teams facing rapidly changing requirements. That being said, XP is the approach to be used when the software requirements are changing dynamically. XP is also set up for project risks such as specific date for the system to be ready, or a completely new challenge for the software team. (Wells, 1999)

XP helps the software development team to (Highsmith and Cockburn, 2001):

- Produce the first delivery in weeks so the team can get fast feedback
- Invent simple solutions so the possible changes are minimal and easier to make
- Test constantly to gain earlier and less expensive defect detection

2.2.4 Lean

Lean software development is an iteration methodology. Lean is value based method that focuses on giving the customer an efficient ”value stream” mechanism which delivers the value to the project. (Gonçalves, 2019)

The main principles of Lean are (Gonçalves, 2019):

- Eliminate waste
- Amplify learning
- Make decisions as late as possible
- Deliver results as quickly as possible
- Empower the team
- Build integrity
- Envision the whole project

2.3 Agile Modeling (AM)

Agile Modeling (AM) is a practice-based methodology that describes how to use modeling and documentation of software systems effectively. So basically, AM helps to find out the line between documenting the system effectively, yet not so much that it slows down the project. The techniques of AM are meant to be applied with agile approaches such as XP or Scrum. It is important to note that AM is not a complete software process, rather it is a collection of practices that are guided by principles and values. In other words, AM is not a complete methodology, it’s a supplement to existing methods, with a main focus on effective modeling and secondary focus on documentation. There are two main reasons why modeling is needed: to get better understanding of the system you are building and the different aspects of it, and to help you to communicate within your team or with your project stakeholders. (Ambler, 2002)

In figure 2 we can see the relationships between models, documents, documentation, and source code. From the point of view of Agile Modeling, a document is any artifact external to source code, which purpose is to contain information. A model on the other hand is an abstraction that describes aspects of a problem or a potential solution to a problem. Some

models will become documents or part of documents, although many of them will be discarded after they have fulfilled their purpose. Source code is a sequence of instructions, including comments. In this context the term documentation includes both documents and comments in the source code. (Ambler, 2002)

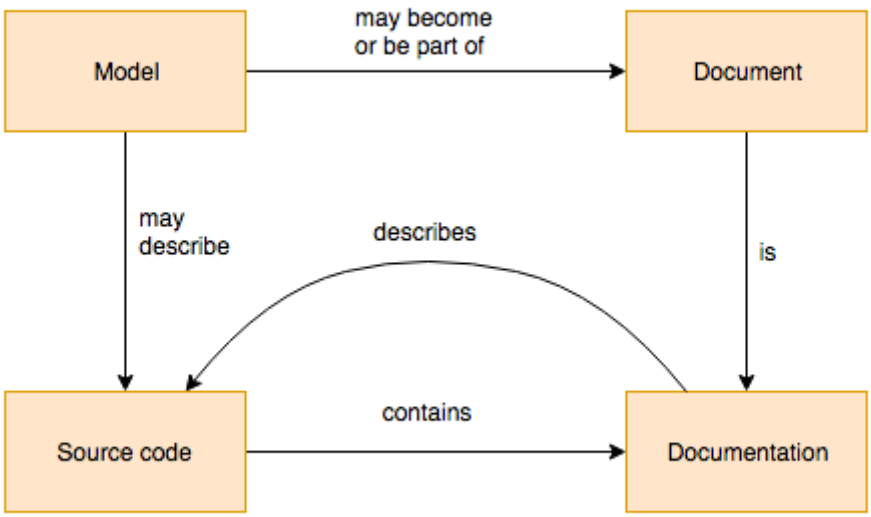

Figure 2. The relationships between models, documents, documentation, and source code (Ambler, 2002)

2.4 The Scaled Agile Framework® (SAFe)

The Scaled Agile Framework (SAFe) is a framework for applying Lean and Agile practices at enterprise scale (Mehrabian, 2013). Since 2011, when the framework was formed, the benefits of SAFe including: faster time-to-market, increase in productivity and quality, and more motivated employees, have been recognized by hundreds of the world's largest enterprises (Scaled Agile, Inc., 2019; Scaled Agile, Inc., 2018). SAFe works by summing up considered leadership from Lean systems thinking, product development flow, Scrum, Kanban, and team building into practices that can fulfill the needs of stakeholders. (Scaled Agile, Inc., 2018) SAFe is provided by Scaled Agile, Inc. where their core belief according to Leffingwell et al. (2018) is: "Better systems and software make the world a better place". Leffingwell points out that the case studies show that many enterprises whether they were small or large have gotten remarkable business benefits from applying the framework. Usually SAFe Agile teams use a blend of agile methods and practices such as Scrum, Kanban

and XP. Agile teams apply small user stories and base their work on short iterations. Teams work towards the next iteration and meet daily so they can reach the iteration goals. At the end of the iteration, teams demo the working system and try to figure out how to improve the process if needed. (Scaled Agile, Inc., 2018)

3 DOCUMENTATION

Documentation is a process for recording information that is produced during the life cycle of a software development process. Documentation usually includes activities such as planning, design, producing, editing, distributing, and maintaining software documentation. (Rico et al., 2009) Software documentation is an important part of software engineering and its projects. However, in many cases documentation is outdated and irrelevant. The role of documentation in a software engineering environment is to communicate information about the software system. (Forward, 2002) Still software documentation is usually practiced poorly and incompetently in most of the organizations because of various reasons, such as the absence of technical writers on the team, the opinion that documentation is an optional task and it should be done on the "free time", the difficulty to document large systems, but most of all because it's not easy to write and maintain documentation. (Steinberger and Prakash, 2011) Some typical problems with documentation in addition to nonexistent or poor quality, and out-dated documentation are difficulty to access the documentation, and lack of interest from the programmers (De Souza et al., 2005).

Often the problem with documentation is the lack of it, but when produced, documents tend to not follow the standards and they might not be understandable or even usable. (Briand, 2003; Forward, 2002) Documents are also rarely maintained. Although, in real software environment where all the other factors along with documentation must be considered, maintaining documentation might not be the most beneficial and suitable approach. (Forward, 2002) However, documentation don't have to mean an actual document file filled with text, rather documentation can have many forms. Like (Forward and Lethbridge, 2002) stated: "A software document may be described as any artifact intended to communicate information on the software system". In addition to this, (Stettina and Kroon, 2013) found out that different artifacts are useful for different purposes and at different stages, for example user documentation is a good way to learn about the environment, and design artifacts and design decisions help to learn about the system and architecture.

3.1 Different Types of Documents

Software documentation can be divided into product documentation and process documentation like figure 3 shows. Process documentation records the development and maintenance process and it is produced to help manage the development of the system. Product documentation on the other hand concentrates on the product and it is typically used when the system is operational. Product documentation is also crucial for management of the system development. (Sommerville, 2001) This thesis concentrates only on product documentation and therefore process documentation, which includes for example project plans, is not presented.

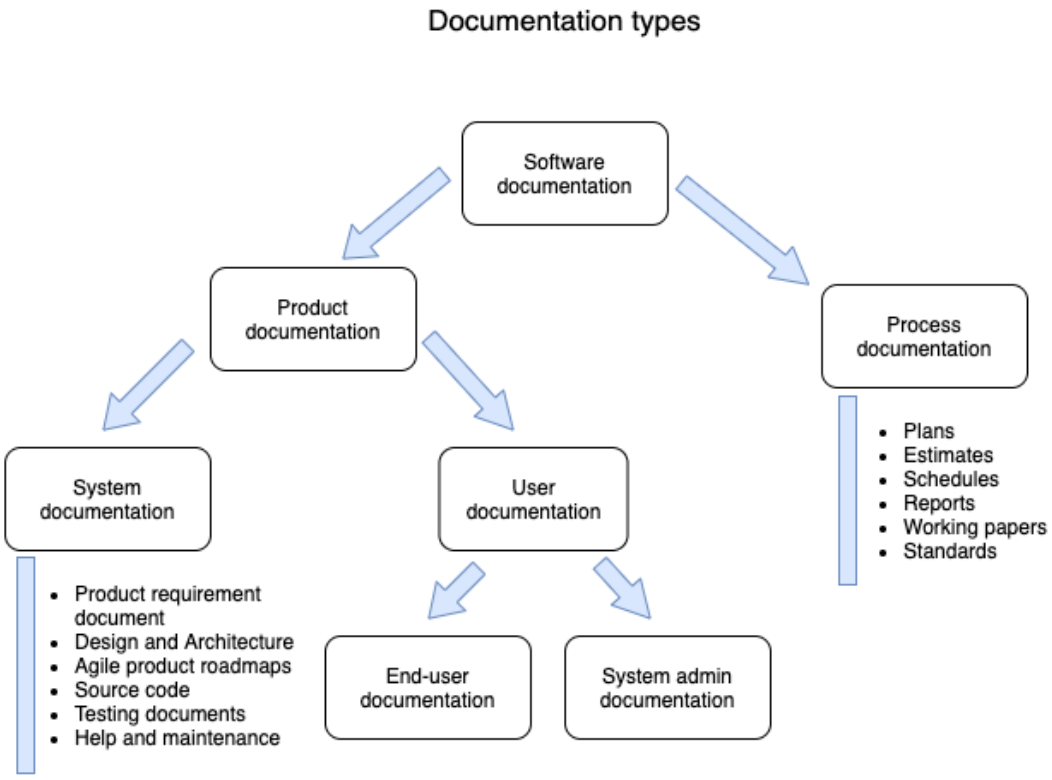

Figure 3. Common documentation types used in Agile projects (Altexsoft, 2019)

Product Documentation

Product documentation describes the software product that is being developed. Product documentation includes user documentation and system documentation. User documentation provides users a description of how to use the software product. System

documentation describes the product more from point of view of developers and maintenance engineers. (Sommerville, 2001)

System documentation

System documentation provides an overview of the system and helps engineers (and stakeholders) to understand the technology behind the system. (Altexsoft, 2019) System documentation includes all the documents describing the system such as requirements documents, architecture design, source code, validation documents, verification and testing information, and system maintenance or help guide. (Altexsoft, 2019; Sommerville, 2001)

- **Requirements specification document:**

  The requirements for a system describe what the system should do, which services it provides, and which are its operational limitations. These requirements reflect customer needs for a system. (Sommerville, 2011) The requirements statements in the requirements document are usually grouped into functional and non-functional requirements. (Maciaszek, 2007) However, adapting to changing requirements is one of the principles of agile approaches (Paetsch et al., 2003). There are both internal and external reasons for changing requirements. Some of the most common reasons are that the complexity of requirements is underestimated (internal), the change of infrastructure or architecture (internal), customers are not sure what they need at the beginning (external), and that the customers change their mind about what they require or requirements are misunderstood (external). (Hotomski et al., 2016) In addition to possibly changing requirements, another problem related to requirements specification is ambiguous requirements and the uncertainty about the status of the requirement. One proposed solution to the problems related to documenting requirements is to specify some rules. Firstly, there should be a definition, which defines what is the required minimum accuracy of the requirements. Secondly, there should be a definition of the readiness of the requirement ("definition of ready"), which defines common minimum criteria for the approval of the features. In addition to these, the crucial requirements should have a defined acceptance criteria, which will tell which criteria must be fulfilled before the requirement can be stated accomplished. (Lehtonen et al., 2014)

- **Architectural design:**

  Architectural design identifies system's main structural components and the relationships between them. It can also be seen as a link between the design and requirements. The produced documentation is usually architectural model that describes how the system is organized. (Sommerville, 2011) Although, in order to understand the architecture, it's also important to document the key architecture decisions (Tyree and Akerman, 2005). It's not necessary to list every architectural decision, rather the most relevant and challenging ones would be desirable to list (Altexsoft, 2019).

- **Product roadmap:**

  In Agile software development processes the strategy and goals of the project are usually documented with product roadmaps. Product roadmap can represent high-level objectives, low-level details, or for example prioritization of tasks. Product roadmap can also be a part of process documentation. (Altexsoft, 2019)

- **Source code:**

  From the point of view of software maintenance, source code and comments are the most important artifacts in order to understand the system (De Souza et al., 2005). A good way to comment the code is to write to clarify some aspects of the code that are not so obvious. For software organizations, a good way to ensure proper internal documentation is by formulating coding standards and coding guidelines. Although, even if the code is carefully commented, it has been found out that meaningful variable names are most useful in understanding a piece of code. (Mall, 2018)

- **Testing document:**

  Different types of testing documents are produced during software development processes. Some of the most common testing documents produced during Agile projects are quality management plan, test strategy, test plan, test case specifications, and test checklists. (Altexsoft, 2019) In addition to documenting the actual tests, the rationale behind the test should also be documented in order to prevent the situations where it's unclear, which aspects have been tested. (Turk et al., 2005)

- **Help guides and maintenance:**

  The documentation related to maintenance and help guides should describe common problems related to the system and provide solutions to them. These documents can also describe the correlations between different parts of the system. (Altexsoft, 2019)

User documentation

User documentation is created for the users of the system. (Altexsoft, 2019) Each system has different types of users with different needs and the users of a system can roughly be structured into two main categories: end-users and system administrators. (Sommerville, 2001) User documentation is usually in a form of manuals, which can be for example tutorials, user guides, troubleshooting manuals, installation manuals, reference manuals, or system administrators guides. (Altexsoft, 2019; Sommerville, 2001) The documentation that is created for end-users should explain how the software can help to solve the problems they are dealing with and this should be explained in the shortest way possible. On the other hand, the documentation targeted to system administrators should provide information about how to operate the software. Usually administration documents cover installation and updates which help with product maintenance. (Altexsoft, 2019)

The users of a software system need to know how to use and interact with the system and that is why they need user documentation. The audience, the purpose, and the form are the main elements of user documentation. It is very important that user documentation fits for the user's needs and that is why audience analysis has to be done. There are differences between the user groups, and it is really important to know the audience, to know who the users are, because documentation probably won't fit for different users' needs at the same time. User documentation has to be produced in a way that it serves the needs of the target users. The importance of the purpose behind the document lies in the fact that the user needs to know how to accomplish certain tasks related to software system or how to act in case of errors. The form of the documentation on the other hand is closely related to both the audience and the purpose. There are many possible forms of documentation, some of which are more traditional than others such as manuals and guides. (Gastegger and Zünd, 2015)

3.2 Agile Documentation

When agile methods are considered, it is a quite common opinion that only a little documentation is required. On the other hand, some people think that agile methods don't produce enough documentation. That brings us to the key question: How much documentation is enough? However, all agile approaches should include at least some amount of documentation so in the end it is the responsibility of the development team to make sure that there is enough documentation available for future maintenance (Paetsch et al., 2003). As Highsmith and Cockburn (2001) state: "Agility, ultimately, is about creating and responding to change." According to Rubin and Rubin agile methods require less documentation for tasks and promote the collaboration between system stakeholders. Traditional software engineering methods emphasize careful planning and design while agile methods concentrate more on the actual software implementation. However, it is important to note that agile development methods do not prevent the use of documentation, rather when compared to traditional software development, agile software development is less documentation-oriented. (Rubin and Rubin, 2010)

Since documentation is sometimes compromised while using agile methods, some of the important knowledge may be lost during and after system development process. Although agile methods overcome the scarcity of documentation by relying on constant collaboration between developers and users. If documentation is adaptive and it supports the collaboration between people rather than replacing it, then documentation can be well placed with agile development principles. However, as noted, while traditional development methods might tend to over-document, an enormous weakness of agile methods is insufficient documentation. (Rubin and Rubin, 2010) It is claimed by agile process community that by using informal personal communication more is gained than by using communication that is based on formal documentation, even though it leads to the situation where tracking information is not possible. However, an advantage of face-to-face communication is that people involved can affect the direction of discussion if they need information related to certain topic. (Turk et al., 2005)

Usually the level of agility that can be reached is related to the size of the development team. Limited documentation is possible in small teams, however, when the size of the team grows, more documentation is needed. As the team grows, the importance of documentation grows because direct communication between the team and the customer might not be possible with the large team, and in these cases documentation usually becomes a way to share the knowledge. (Sillitti and Succi, 2005) Ambler (2016) offers a way to logically decide if you actually need to create a certain document. This logic is presented in figure 4.

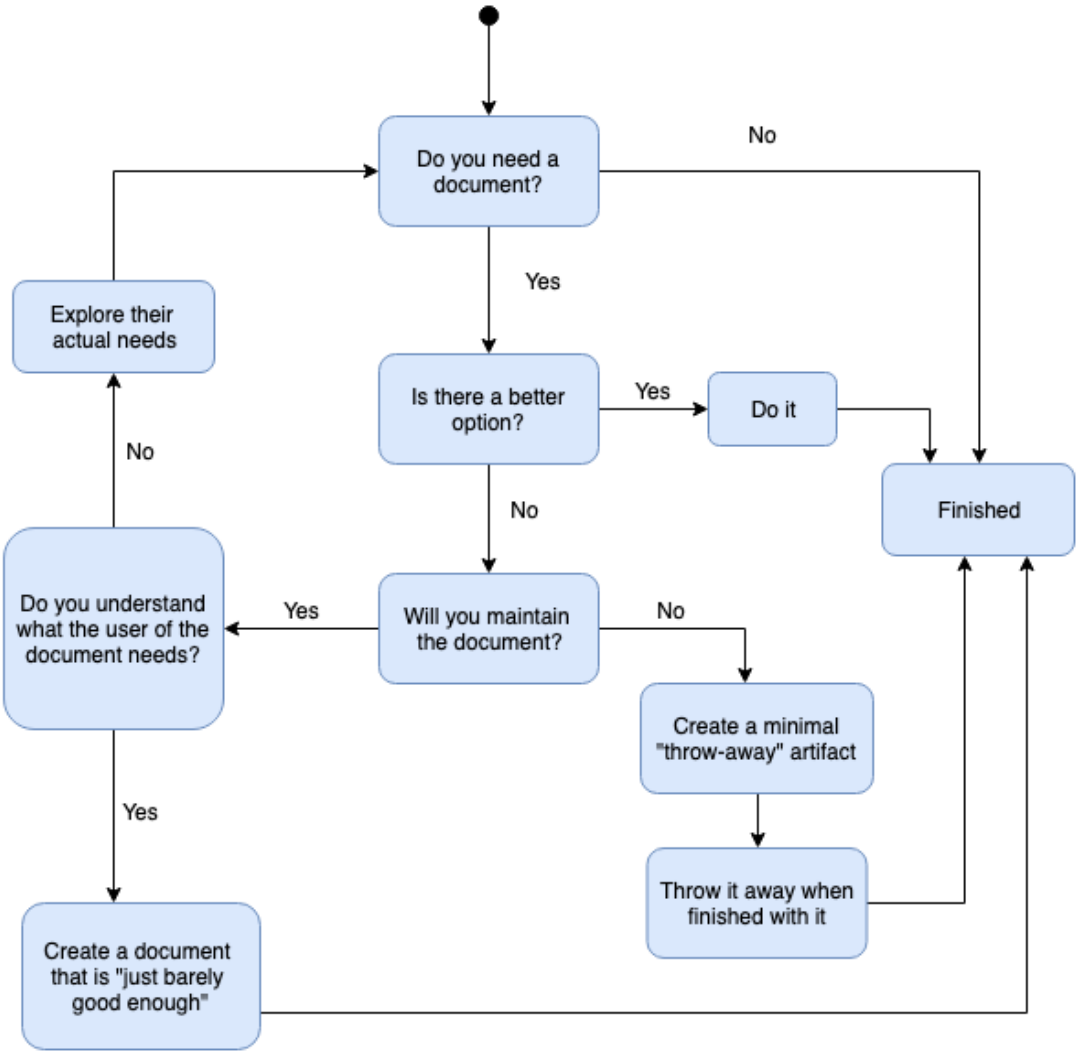

Figure 4. Flowchart that helps to decide if creating documentation is actually needed (Ambler, 2016)

It can be seen from the figure 4 that there are basically five different solutions. The first solution is that documentation is not needed. The second option is that documentation is needed, but there is a better option than documentation available. The third solution is that

documentation is needed but the document won't be maintained and therefore some temporary documentation artifact should be created instead of an actual thorough document. The fourth solution is that documentation is needed, and document should be created. The fifth and final solution is that documentation is needed, but user needs for the documentation are not understood so the user needs should be explored more closely.

3.3 When is a document agile?

According to Ambler document is agile when it meets the following criteria: (Ambler, 2002; Ambler, 2005)

- **Agile documents maximize stakeholder return of investment (ROI):** Agile document should provide a benefit that is greater than the investment in the creation and maintenance of the document. So basically, the documentation should at least provide positive value.

- **Stakeholders know the total cost of ownership (TCO) of the document:** Stakeholders need to understand the total cost of ownership (TCO) of the document and they need to be willing to invest in the creation and maintenance of the document.

- **Agile documents are lean and sufficient:** When a document is agile it contains just enough information to fulfill its purpose. An agile document is as simple as it can possibly be. For example, agile document can capture the critical information without investing time to make it look "pretty".

- **Agile documents fulfill a purpose:** There should always be a purpose for the creation of the document, and an agile document should fulfill this defined purpose.

- **Agile documents describe information that is less likely to change:** If there is a great chance that the information will change, there is not much value to invest time writing (especially external) documentation, because the information might change before you are even finished writing it, and it can also make it more difficult to maintain the documentation.

- **Agile documents describe "good things to know":** Agile documents capture critical information and leave out the obvious information. The critical information can for example be a design rationale, requirements, or operational procedures.

- **Agile documents have a specific customer and facilitate the work efforts of that customer:** Usually there are different types of customers and users of the system, which means that probably different types of documents and different writing styles are needed. For example, system documentation is usually written for maintenance developers to provide an overview of the system's architecture, and it can also summarize the critical requirements and design decisions. User documentation on the other hand often includes tutorials for using the system and is written in a way that it's understandable for the users. Operations documentation usually describes how to run the system and is written in a way that operations staff understand it.

- **Agile documents are sufficiently accurate, consistent, and detailed:** Agile documents don't have to be perfect, more importantly they just need to be good enough. For example, you can learn how to use a new software by using a book that describes the previous version of the software, although this probably won't be a perfect situation and it won't cover all the features, but it still helps you to get the software up and running. However, it should be noted that this depends on the system. If the documentation is produced for software system for nuclear power plant, the documentation should be accurate and up-to-date.

3.4 Why do people document?

Selic (2009) points out that documentation is rarely created to benefit the author of the documentation rather the purpose is to help others. According to Selic a common reason for documentation is to instruct the people who are unfamiliar with the system or have forgotten it. This can be done by communicating information related to how the system is structured, how it works, and what is the design rationale behind the system. Selic also points out that sometimes explaining things to others helps to clarify the author's own thoughts too.

Feduniak (2016) points out some main reasons why documentation is needed:

- Stakeholders require documentation
- Customers want certainty
- To keep track of every aspect of the project
- For audit purposes
- For training material for new team members

Ambler has quite similar thoughts about the issue of why people should create documentation. According to Ambler there are six valid reasons to create documentation: (Ambler, 2002; Ambler, 2005)

1. **Your project stakeholder requires it**

   Fundamentally the creation of documentation is a business decision, not technical. Since you are investing the resources of your project stakeholders in the development of the documentation therefore it should be up to them to decide whether the money is spent that way or not. If your project stakeholders request a document from you, understanding the trade-offs involved, then you must create the document. Project stakeholders are a collection of people including all of the clients of the system, maintenance developers, users and their management, and operations staff. However, they will all request different types of documentation.

2. **To define a contract model**

   Contract model defines how your system and an external system interact with one another. Usually contract models are required when an external group controls information resource that your system requires, for example database or information service.

3. **To support communication with an external group**

   Sometimes you need to work with an external group of people because it is not always possible to co-locate a development team or have all of the project stakeholders available at all times. That is why you need to find ways to communicate with the external group of people and shared documentation is often part of the

solution. Even though it is a mistake to use documentation as a primary way of communication, since written information can easily cause misunderstandings, it is a good supporting mechanism.

4. **To support organizational memory**

   It is important to note that it is not enough to just develop the software, but it is also needed to develop the supporting documentation that is required to use, operate, support, and maintain the software over the time.

5. **For audit purposes**

   In some fields you have to follow a defined process and you also need to have proof that you did so, which results in more documentation than you would have probably normally been written.

6. **To think something through**

   Some people write a documentation to simply increase their own understanding. Writing down your ideas can help you to concrete them and also help you to discover problems with your thinking. What appears clean and simple in your mind can often turn out to be very complicated once you try to describe it in detail, and that is where you can benefit from writing it down first.

Lethbridge et al. (2003) found out that documentation was rated effective or extremely effective in order to complete the following tasks: learning a software system, testing a software system, working with a new software system, and solving problems when other developers are unavailable to answer questions (Lethbridge et al., 2003). Turk et al. (2002) point out that good documentation of requirements and design is crucial to ensure that all of the team members share the same vision of the product throughout the development process. However, they also note that documentation and models should only be created and maintained if they add value to the project and stakeholders. VanAlbrecht and Nemani (2014) point out that without consideration of who will use the documented information and for what purpose, the benefit compared to the effort is hard to determine.

Wagenaar et al. (2018) point out five major elements behind the rationale for the usage of agile documentation artifacts:

1) Documentation artifacts might come as prescribed in agile software development

2) Documentation artifacts provide useful governance for the team

3) Documentation artifacts are useful and/or necessary for internal communication which means that the communication then don't happen face-to-face

4) Documentation artifacts are useful and/or necessary because of quality reasons, and

5) External parties need the documentation artifacts.

In many cases the reason to create documentation seem to be one of the following: stakeholder requires a document, audit purposes, internal communication, tracking the different aspects of the project, source of information for external parties, confirmation that all the team members share the same vision of the product, support of organizational memory, or simply a way to think something through.

3.5 How to produce useful and effective documentation?

Effective documentation is an act of balancing between having just enough documentation at just the right time for just the right audience. Ambler thinks that to accomplish this balance, the following issues must be considered: (Ambler, 2002; Ambler, 2005)

- **Software development vs documentation development:** This refers to the issue that any time spent creating documentation is a time not spent developing the software. The primary goal (as it is stated also in agile manifesto) is to produce working software. However, the secondary goal is to enable the next effort and therefore writing a documentation is still needed.

- **Executable specifications offer far more value than static documentation:** Executable specifications, which can be for example customer test suite or a developer test suite offer a huge value to developers since in addition to specifying

the system they also help to validate it. A customer test suite specifies the majority of the requirements and a developer test suite specifies the detailed design.

- **Software developers have the knowledge, technical writers have the skill:** Usually only few technical people actually have good writing skills. The problem with this is that naturally the best person for writing documentation is the one familiar with the topic, for example when considering a system being developed it would be the developers of that system. There is basically three approaches how to handle the production of documentation: the technical writer will handle the documentation, the developer will write the initial version of documentation and then the technical writer will finish and clean it up, or the technical writer and the developer will work together to write the documentation.

- **What is required during development is often different than what is required after development:** The needs are different during the development process and after the process. During the development the problem and the solution are explored by trying to figure out what is needed to be built and how things work together. On the other hand, after the development it is more important to understand what was built, why it was built in a certain way, and how to operate it. Sketches and drafts are typically made during the development and that is usually fine since it helps to travel light. However, more formal documentation is often wanted after the development.

- **Willingness to write documentation vs willingness to read it:** Sometimes there are situations that people ask questions related to something that is clearly documented, and they might even have the document lying at their desk. The issue is that you can write documentation, but it doesn't guarantee that people will read it, and this is why it is important to consider how much to write.

- **Do you document as you work or when you are finished?:** There is basically two "extremes" of when to write the documentation. One extreme is to write all of the documentation together with developing the software. The advantage of this approach is that you are capturing information as you progress although the

disadvantage is that as the software evolves, the documentation will need rewriting, which slows down the development process and results in wasted effort. The other extreme is to wait until the development process is finished and write the documentation afterwards. The advantage of this approach is that you are writing about a stable information. However, there are also disadvantages in this approach such as: forgetting some of the reasons behind the decisions that have been made, the "right" people might not be available to write the documentation anymore, or you may not even have the funding to write the documentation anymore, let alone willingness to write it. As a conclusion, the effective middle ground is to capture the information during the project as it stabilizes.

- **Can you wait until the information has stabilized?:** It might not be beneficial to invest time on documenting speculative ideas such as the requirements or design early in a project because if the written documentation contains information that hasn't stabilized yet, the documentation will probably need to be reworked after the information has changed. Although, if the documentation is produced later in the life cycle when the information has stabilized it will most probably result that the documentation effort is few iterations behind the software development effort.

- **Code documentation vs external documentation:** There are choices to be made related to documentation, such as whether to place the documentation in the code or to place it in external artifacts. Again, this is a situation where an effective middle ground is needed to be found. Whether the documentation should be internal documentation as code comment or external documentation as supporting artifacts depends on the audience. There is an AM practice, Single Source Information, which suggests that you should try to capture information only once and in the best place possible.

As a summary of Ambler's thoughts, the most important things to consider about production of the documentation are: the time spent writing documentation, who should write it, what information is required in which phase of the development and has the information stabilized, when to write the documentation, and where to place the documentation.

Feduniak (2016) also points out some good practices for agile documentation:

- **Document only the relevant information:** When using agile methods, only the most essential information should be documented. For example, decisions should only be documented if there are alternatives. Overlapping should be minimized and information shouldn't be repeated. Also, executable specifications should be preferred over theoretical ideas.

- **Wait before documenting:** The best way to avoid incorrect information is to document late. If you wait until the decisions have been implemented the information is stable and shouldn't change anymore. When you document late, rewriting can be minimized.

- **Get a technical writer:** The advantage to get a technical writer to write the documentation is that there is someone who is in charge of documentation. Also, the documentation will most probably be done well. Developers are usually great in technical area, but when it comes to describing things to others, it might not be easy.

- **Keep the documentation in one place:** Documents should be accessible, transparent and available for everyone who needs them. That is why documentation should be organized well and most preferably in one place. A good way to do this is to use wiki as a documentation tool.

Also Scott (2016) offers some guidelines for creating useful documentation:

- **Identify the audience:** Identifying the audience is extremely important in order to write for example useful user stories. If the audience is identified, the user story can be written in a way that it provides the information about the target audience, without having to specify it separately.

- **Include the writer on the team:** If there is a specified writer of the documentation, it helps if the writer works with the team. In that way writer can ask questions and get answers as the development is happening.

- **Don't focus on the final documentation:** In agile processes documentation shouldn't be planned far ahead because the changes might happen in any phase of the development. When agile methods are used, the good way is to deliver small pieces of documentation during the development, and not to keep the focus on final documentation set. If smaller chunks of documentation have been produced during the development, it is easier to put the pieces together at the end.

According to Hanssen et al. (2018) one of the important things related to the documentation in an agile way is the reusability of the documents. Because of aiming to less documentation, combining documents is also a good way to reduce the amount of documentation. From the point of view of agility, the best solution might be automatically generated documents since many of the software engineers want to write the code, not documentation. The great thing about reusable documents is that the extra costs are low. Reusable document can be said to be a document where large parts are used as they are, while only small parts need to be adapted for each project or sprint. Also, if reuse is set as a goal at the start, the changes between projects or iterations won't be big. (Hanssen et al., 2018) In order to produce useful and effective documentation, there are also a lot of other things to consider such as the tools, techniques, style and format that is being used during the documentation process. However, as Hanssen et al. (2018) point out, documentation might have different forms but it's the content that matters, not the format.

3.6 Which documentation tools and techniques should be used?

Tools and techniques used in a development process are usually defined by the documentation type and documented information as well as the target audience of the particular document.

Unified Modeling Language (UML)

One documentation technique suggested by De Lucia and Qusef (2010) is to use computer-based tools like Unified Modeling Language (UML) and project management tools to create the high level description of the project, and to document for example requirements Also Ersoy and Mahdy (2015) point out UML as a documentation technique to help minimize the documentation. UML-based documentation is usually created during the agile development process, and it is believed to derive team motivation and easy updates, although in the long run it lowers the sustainability of the knowledge sharing documentation (Ersoy and Mahdy, 2015). Rehman et al. (2018) point out that minimal and simplified documentation in the form of use cases and test cases increases the maintainability and traceability of a software system. Hess et al. (2017) aim to form documentation guidelines for agile teams and they discovered user stories to be the central RE artifacts in the agile projects. Further requirements information was usually specified with the help of Personas or usage scenarios. Overall, to derive architectural decisions user stories and usage scenarios were considered as the most important artifacts. To derive interaction and visual user interface (UI) design related decisions, all these artifacts (epics, personas, user stories, and usage scenarios) were considered important, but the usage scenarios were considered as the most important by the usability engineers. For preparing and running system test cases, none of the earlier mentioned artifacts were considered very important. Usage scenarios were considered with the highest relevance, then user stories, and then epics and personas with the same level of importance. (Hess et al., 2017)

User Stories

As pointed out above, Hess et al. (2017) found user stories to be a good way to document requirements. User stories are presented as a short and simple descriptions of a feature, and they are usually described from the customer perspective. However, user stories don't replace requirements documents, rather they are a good addition to them, and they can for example be pointers to the real requirements or certain diagrams. (Cohn, 2019; Ibanez, 2018) Ibanez (2018) points out that usually requirements documentation concentrates too much on the keywords *who* and *what*, even though the keyword *why* is extremely important in order to understand the relevance of requirements.

Wiki

Ersoy and Mahdy (2015) point out the use of Wiki as a good documentation technique since it is the most revised social software development tool in terms of usage and features, and it is also suitable for both small teams and large enterprises. They also mention Semantic Wiki, because it is a lightweight model which classifies concepts. It is also noted that using mockups as a Wiki documentation technique is a good approach because "a picture is worth a thousand words" and it helps to keep the website visual. (Ersoy and Mahdy, 2015) After taking a closer look at the artifacts used in agile projects, Voigt et al. (2016) suggest that every artifact that is used can be considered as documentation. Usually simple tools such as note cards and whiteboards are used to record information temporarily during agile processes and Wikis have been identified as the best electronic tool for that. There are different types of wikis like semantic, structured, and hybrid wikis, which enable users to manage the information and also add structures to them. (Voigt et al., 2016)

Voigt et al. (2016) also mention other tools that are used in agile processes, which are: Integrated Development Environments (IDE), Version Control Systems (VCS), and issue trackers. It is also mentioned that information has to be traceable between different tools. These tools are focusing on different things in the development process: the issue tracker and wiki focus on the functional level while the IDE and VCS focus on the source-code. VCS and issue tracker manage and track changes, and IDE and wiki focus on the software that is being produced. (Voigt et al., 2016) Robillard et al. (2017) point out that in the end documentation tools offer quite a little help to creating the actual content of the documentation. However, they also mention that web-based collaboration platforms have enabled new opportunities for the creation of documentation, and also how to access them.

3.7 When should you create documentation?

As it has been pointed out earlier, there are two extremes of when to create documentation. One approach is to create documentation after the development process when the information is stable, and the other approach is to create documentation during the development process. These two approaches can also be referred as *document late strategy* and *document continuously strategy*.

Writing the documentation in the software development life cycle (SDLC) differs in traditional and agile development. In figure 5 we can see the document late strategy. The curves represent the total amount of effort that is invested in writing documentation. In document late strategy some of the documentation artifacts such as project plans and specifications might not be retained at the end of the project, although in some cases they will be (Ambler; 2002, 2005).

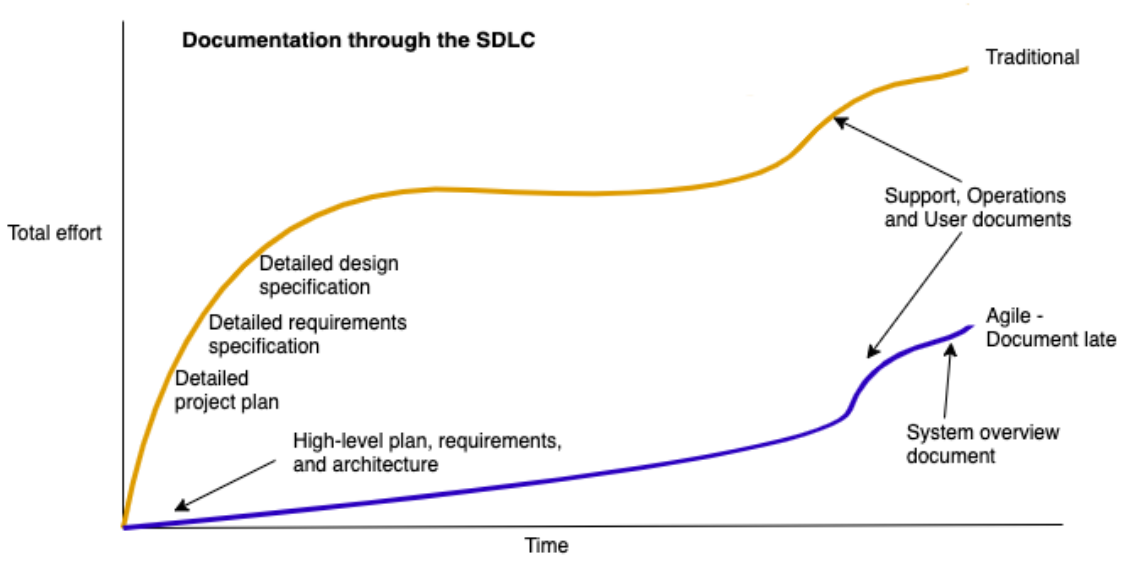

Figure 5. Documentation throughout the software development lifecycle - Document late strategy - according to Ambler (2005)

Figure 6 shows the document continuously strategy. In this approach documents such as support documents, operations, system overview, and user documents are produced throughout the development process. The idea behind this approach is that if the system is to be potentially shippable at the end of each iteration then the deliverable documents are too. The challenge is that all documentation created needs to be evolved in sync with the code during the process. (Ambler; 2002, 2005)

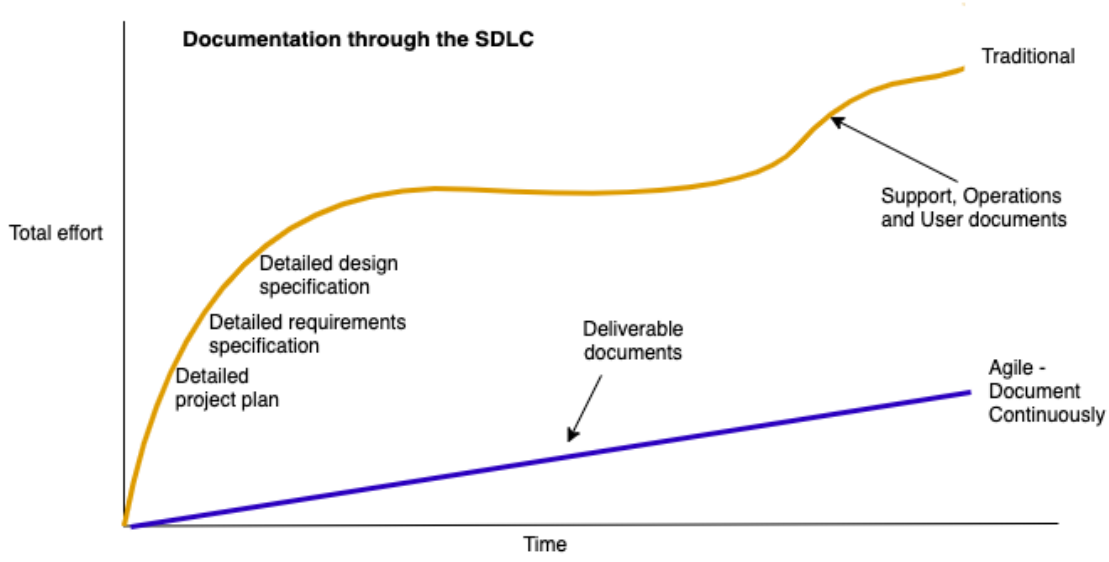

Figure 6. Documentation throughout the software development lifecycle - Document continuously strategy - according to Ambler (2005)

Stettina and Heijstek (2011) came to the conclusion that documentation which is in text format benefits from being produced iteratively. By that note it could be said that textual documentation fits better in the document continuously strategy. However, Stettina and Heijstek also point out that documents with diagrams such as UML models are considered easier to update than textual documents (Stettina and Heijstek, 2011). Updating and maintaining documentation is also needed to be considered when thinking about when to produce documentation, and it should be noted that updating or rewriting documentation is definitely a part of document continuously strategy.

As "responding to changes" is one of the key principles of agile methods, theoretically customers can request changes in requirements at any time regardless the stage of the project (Hoda et al., 2012). De Lucia and Qusef offer a guideline "no early documentation", since documentation that is produced in the early stages can become irrelevant really quickly because of the possibly changing requirements (De Lucia and Qusef, 2010). On the other hand, according to Rehman et al. (2018) documentation update should be made after each phase, because by documenting the iteration the unity of the system will be maintained.

Wagenaar et al. (2018) point out that different documentation artifacts should be produced in different phases of the development process. Table 1 shows in which development phase should certain information be documented according to Wagenaar et al.

Table 1. Produced documentation artifacts and in which phase of the development process they are created according to Wagenaar et al. (2018)

Pre-development	**Development**	**Post-development**
Requirements	*Acceptance criteria*	*Implementation guides*
Product requirements	*Architecture standard*	*Releases*
Business cases	*Bug reports*	*Release checklists*
Market requirements	*Definition of done*	*User manuals*
	Definition of ready	
	Functional design	
	Source code	
	Technical design	
	Technical documentation	
	Technical requirements	
	Test cases	
	Test reports	
	Use cases	
	User stories	

From table 1 it can be seen that most of these documentation artifacts should be produced during the development process. Cohn (2019) also points out that user stories should be created during development in agile software development processes.

3.8 Who should produce the documentation?

VanAlbrecht and Nemani (2014) point out that it's important to consider the author of the documentation. They point out that the issue of who will write the documentation should be questioned, and even though the team roles might be specified, or documentation is assigned as a task for some specific role, in agile projects this might not be the best way to do things. For example, Forward (2002) points out that usually a huge amount of the knowledge about the software system is in software engineers' minds and that is why it can affect the quality of documentation negatively if someone else is responsible for writing the documentation. Cohn (2019) points out that basically anyone can write the user stories in the project, and that in "good" agile projects user story examples should be written by every team member. Cohn also states that it is more important to consider who are involved in the discussion of

user stories than who actually writes them. Ambler (2002) notes that it's required to work with the customer in order to create documentation that will fulfil the needs of the customer. If the customer is not involved in the process there is a risk of creating too much or unnecessary documentation, which can on the other hand lead to reduction of the agility of the process. As mentioned before, Ambler (2002; 2005), Feduniak (2016), and Scott (2016) point out that including a technical writer in the documentation writing process is also a good option.

4 EMPIRICAL STUDY - INTERVIEWING THE PRODUCT STAKEHOLDERS

The purpose of the interviews related to this master's thesis was to determine the real needs for the documentation of the product from the point of view of the actual users of this documentation. The interviews also helped with the research questions, which was assumed before the execution of the interviews. In the interviews seven questions were asked, three of which are related to the team and the product itself, and the other four which are related to the documentation of the product. In addition to the interviews, every interviewee filled out a survey form related to the product documentation. The survey form can be found from the appendixes as appendix 1.

4.1 Interview questions

The reason behind the interview questions were to get actual data of the needs of different stakeholders of the product. This data is also assumed to help formulate the conclusions of this thesis. Conclusions in this case mean the proposition of how the case company could improve their documentation process. Overall the interview questions were aimed to help to form a suggestion that includes different documentation types categorized and involving the following issues: what is the target audience of the documentation, what is the content of the documentation, and in which form the certain documentation is created.

The interview questions were mostly composed together with the supervisor from the case company to get the most out of the interviews in terms of fulfilling the empirical aspect of this research. The necessary output related to this master's thesis and the interviews was discussed, and the questions were formed based on this discussion in order to get the opinions of the interviewees from these most important topics. Four main topics in the interviews are based on keywords: **why**, **what**, **how** and **who**.

The interview questions are:

1. Describe your team. What is the size of your team? Is the whole team physically co-located?

2. What is your role in the team?

3. Describe the product briefly. How are you involved with the product?

4. Why do you need the documentation of the product?

5. What should be documented related to the product?

6. How should the documentation be produced? Which styles of documentation should be used? Which tools should be used?

7. Who or which team should produce the documentation related to the product? Why?

At first, questions 4 - 7 were formed based on the discussion with the supervisor and the chosen keywords. After that the first three questions were added to get the background information of the interviewees, and to make the connection between the product and the interviewee. The scope of how precise the interviewees' answers should be was also discussed with the supervisor of the case company beforehand. It was pointed out that if the responses would be too general and not at all detailed and focused, they would be quite useless. Keeping that in mind it was necessary to be prepared to ask some leading questions in the interviews.

4.2 Participants

The participants of the interviews were chosen by the supervisor from the case company in a way that the range of different stakeholders of the product would be as wide as possible. The participants are working in different roles and different teams, and are that way involved with the product in different ways. Altogether 15 interviews were held, 14 of which were individual interviews and one of which was a group interview with four participants. Overall there were 18 participants in the interviews. Nine of the participants are from development

teams (product development and software development), one is from software quality assurance (QA) team, three from customer team, one from service operations team, one from project services, one from partner management, and two from sales and marketing. Over all the participants are working on eight different teams. However, in the service operations team's and sales and marketing team's interviews there were "extra" interviewee present who also gave their opinions about the interview questions. In addition to that the test manager had had conversation with the whole QA team before the interview and presented the whole team's opinions and desires. In order to keep the results clearer only the 18 actual interviewees will be referred here.

In table 2 the 18 participants are listed with their teams and job areas. It can be seen that the participants are working on different tasks. Apart from developers, there were not participants with same job area involved in the interviews. Altogether the participants are working on 16 different job areas.

Table 2. List of participants with their job area and team

Participant	Job area	Team
1	Identity and Access Management (IAM)	Sales & Marketing
2	Process consultant	Customer Consultation Service
3	Partner manager	Partner Management
4	Manager	Software Quality Assurance
5	System architect	Customer Consultation Service
6	Manager	Software Development
7	Customer service	Customer Consultation Service
8	Manager	Service Operations
9	Usability architect	Product Development
10	Marketing	Sales & Marketing
11	Implementation consultation	Project Services
12	Product manuals	Product Development
13	Test automation engineer, CI	Product Development
14	Main architect	Product Development
15*	Product developer	Product Development
16*	Software developer	Software Development
17*	Product developer	Product Development
18*	Software architect	Product Development

*Group interview

The amount of time that the participants have worked on software industry also varies. In figure 7 it can be seen that all of the participants have worked on the industry more than one year. Half of the participants have been working on software industry more than 10 years.

Figure 7. The amount of time that the participants have been working on software industry

4.3 Interview execution

Most of the interviews were executed by face-to-face interviews that were held in Kouvola, but few of the interviews were executed via Skype since not all of the interviewees were located at the same place. Any kind of recording devices were not used rather all of the information was gathered by taking notes by the interviewer. The time that the interviews took was from 15 to 60 minutes, but usually an interview lasted around 30 minutes. In some cases, the interviewees had a lot to say and that is why some of the interviews lasted longer.

Like mentioned before, the interviews consisted of two parts: a survey form filled by the interviewee and the actual interview. In the beginning of every interview (except the ones held via Skype) the interviewees were asked to fill the survey form. On Skype interviews the survey form was sent via email and the interviewee were asked to fill the survey and send it back to the interviewer. On face-to-face interviews after the survey was filled, the

actual interview began. The interviews were held in a way that the interviewer asked the questions and let the interviewee answer freely. If there was a need for more precise answer the interviewer asked more detailed questions that lead the conversation to the right tracks. At the end of every interview the interviewees were asked if they have any other comments related to the topic and all of the interviewees had at least something to say, some more than others. Some issues that were brought up at this point were documentation language, product versions, installation instructions, and documentation related to customers and partners. These issues are discussed in the subchapter 6.4.5. After each interview the notes taken by the interviewer were written down in a clearer way for the further analyzation.

4.4 Interview results

In this subchapter, the results of the interviews are discussed. This subchapter concentrates on the questions 4 - 7, which are related to the documentation of the product, and the results are reviewed question by question. In the end of this subchapter, the other issues and opinions related to the topic that came up in the interviews are also reviewed. Since the participants are working with different tasks and they need different things from the documentation of the product, it naturally showed in the responses by large scale of different type of answers.

4.4.1 Why is the documentation of the product needed?

The things that were mentioned after the question *"Why do you need the documentation of the product?"* included: quality assurance, sharing internal knowledge or familiarizing someone with the product, the customer needs and requirements, the areas that need development, bugs, the things related to how the system works, and the marketing aspect. However, the most common answers can be divided into nine categories: "for customers / end users", "to know how the features work", "to set up the system", "for partners", "to solve the problems", "To familiarize new employee with the product, "to know what the version includes", "to get the big picture of the system", and "to understand different possible solutions". For example, the following reasons were mentioned by the interviewees:

- *"So we can set up the systems and solve the problems."*
- *"To convince the partners that the product is applicable for their needs."*
- *"We need to know what the current version of the product includes."*
- *"We need to know how the new features work when the new version is put to use at the customer."*
- *"New developer needs information about previous executions."*
- *"We need to know how to build a solution and understand how it works."*

Figure 8 shows the percentage of how many times the nine most common reasons were mentioned. Some interviewees mentioned more than one of these things and this is counted in. Although only the reasons that were mentioned at least twice (by different participants) are included.

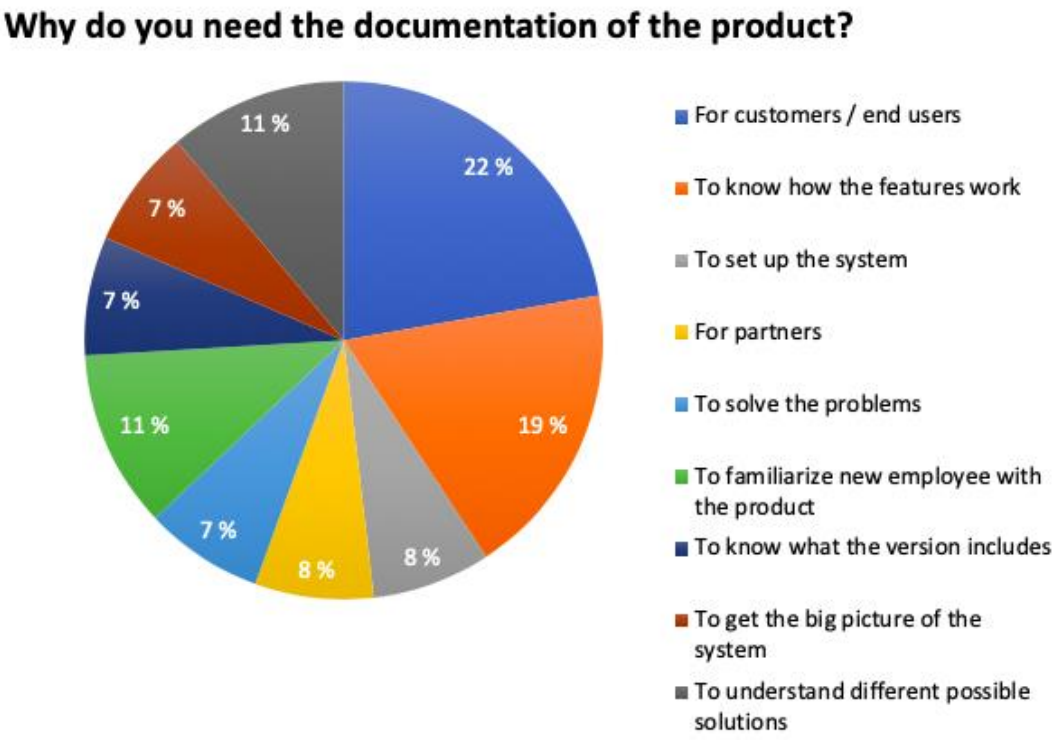

Figure 8. The nine most common reasons for the need of documentation of the product

The reasons "for customers / end users" and "for partners" are classified as external needs for documentation here, and the other four reasons are classified as internal needs for documentation. However, for example the reason "to solve problems" could also be seen as

an external need since often the problem situations appear at the customer end, although in this case this is still seen as an internal need for documentation.

Figure 9 shows that the internal need for the documentation of the product is approximately two thirds (70 %) of the whole need, and that the external need is approximately one third (30 %) of the whole need. However, it should be noted that only the nine most common reasons were included here. It is also important to realize that the responses vary depending on which team the interviewee is in, and for example whether the interviewee is directly involved with the customer or not.

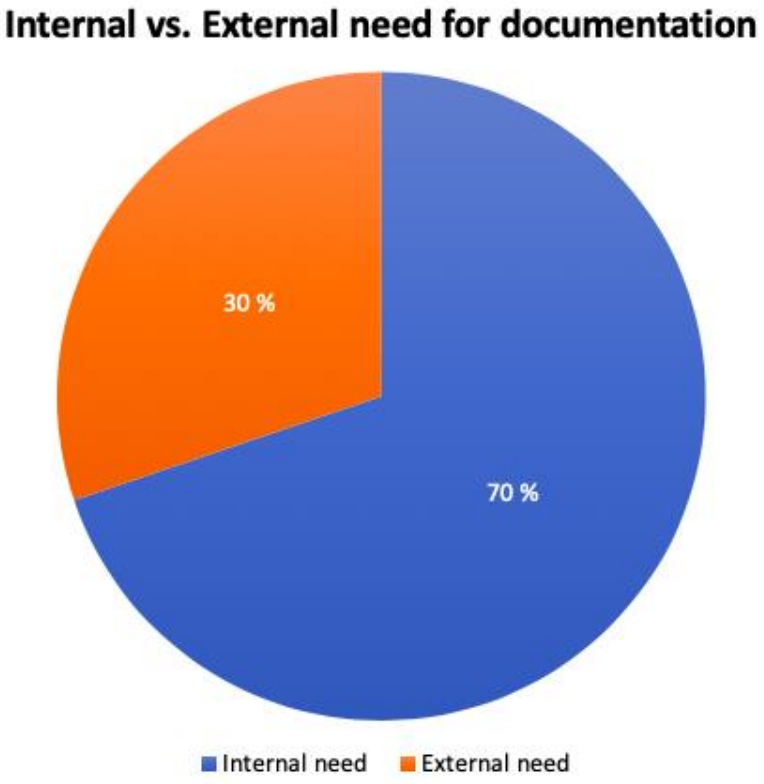

Figure 9. Internal and external need for the documentation of the product

4.4.2 What should be documented?

Since this interview question was *"What should be documented related to the product?"* it gave the participants the freedom to answer basically anything. People working with different tasks need naturally different things from the documentation, but some key lines

were found after the interviews. The most important things to document related to the product seemed to be functionalities, features, requirements, acceptance criteria, interfaces, configurations, modifications, exceptions, errors and bugs, instructions, use cases, database description, and architectural description. Overall there were 13 clear elements that were mentioned during the interviews.

These 13 elements and in how many interviews they were mentioned are shown in figure 10. Although, if the interviewee didn't mention these elements by name it hasn't been counted in. For example, functionalities and features are usually part of the requirement specification process, however if the participant only mentioned the importance of documenting functionalities, then only functionality was counted in as a thing that needs to be documented. As it can be seen from the figure 10, the most important things to document according to the interviews seem to be functionalities, features, and configurations.

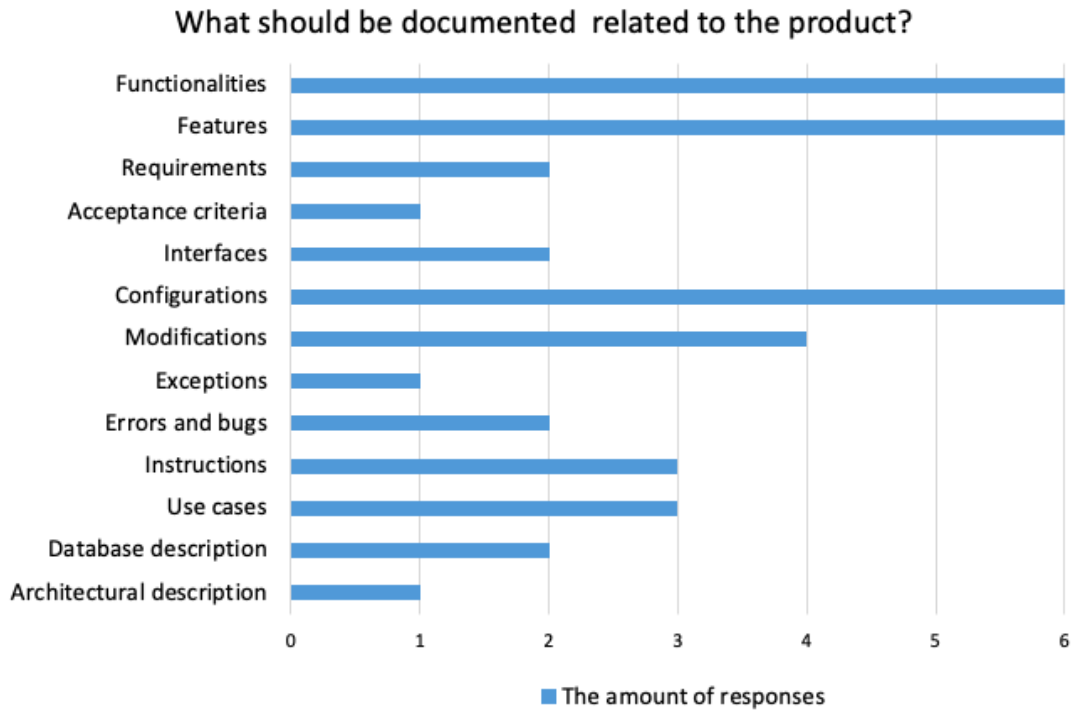

Figure 10. Most important things to document according to the interviews

Below all of the 13 elements listed in figure 10, and the interview results related to them are reviewed. When considering the data from the interviews it is important to note that some of the interviewees might have also meant some of the things that are represented in figure 10

but they might have just said it differently. For example, when considering the question "*What should be documented related to the product?"* and the interviewee answered: *"How it works and why it works that way"*, that basically means the same thing that functionalities and features, even though it still hasn't been counted in. Additionally, many of the things presented in figure 10 are strongly connected to each other, like for example functionalities, features, requirements, and use cases.

Functionalities

It was mentioned in the interviews that up-to-date descriptions of all of the functionalities would be extremely important to have since the understanding of the system lies mainly on the understanding of the functionalities. It was also pointed out that functionalities should have different requirement statuses:

- *"... for example performance related to the loading time of the front page should be extremely short, some other functionality would then have a different requirement status regarding to performance."*

In addition to this, it was mentioned that all new functionalities should have descriptions, and if functionality changes or it is being updated there should be information about how it has changed, how it is supposed to work and overall some kind of announcement that the functionality has been updated. Also instructions of how to take the functionalities into action and information about how the functionalities can be connected to already existing systems are desired. The interviewees are also hoping for detailed configuration descriptions of all the functionalities.

Features

The things that the participants want to know related to features are: what kind of feature it is, how does the feature work, where and how can the feature be used, and what does the feature do. It was pointed out that the descriptions of features shouldn't be "too technical", although they should be exact but still understandable. The assumed configurations related to features were also desired. Some other comments related to features were:

- *"When a feature is taken into action at the customer end, we should have the information about its functionalities and what is required, which configurations are required, how does the feature work, and also information about the technical process."*
- *"The feature is not ready until it has been documented!"*

Requirements and acceptance criteria

Requirements specification seemed to be extremely important to the software quality assurance team. It was pointed out that in order to test things the requirements need to be unambiguous and straightforward. It was also pointed out that non-functional requirements such as performance, maintainability, scalability, usability, availability, and portability should also be included. Requirements should also be classified and prioritized according to how important they are to the customer. It was also mentioned that requirements should have a so-called status:

- *"Currently we don't know in which state the requirements are. Requirements should have a status and it could be for example either draft, approved or reviewed."*

It was also pointed out that it would be important to know the author or owner of the requirement in cases where there is a need to ask questions regarding to a specific requirement. One of the most important things related to requirements was said to be that requirements should include three qualities: who, what, and why. So basically, it's important to know who "created" the requirement, what is required, and why it's required. And even if it might not seem like it, the reason behind the requirement is very important.

One desired thing related to requirements is acceptance criteria. It is hoped that acceptance criteria and "definition of ready" would be taken into use:

- *"We would want that so-called definition of ready would be taken into use. This would be a specification that when the requirement fulfills the criteria of definition of ready it can move into production."*

Basically, the definition of ready would mean that certain things should be done before the implementation can start. About acceptance criteria it was also pointed out that it doesn't even matter how they are presented, and they can be shown for example by bullet points if they at least have information about user type, what should be done, and why it needs to be done. One important thing that was mentioned is that there should be a common vision of requirements specification:

- *"Everybody should have the same vision about requirements specification. Currently requirements might be created late and they might include misunderstandings. Before the implementation, there should be a common vision of what the requirements include. The customer should also be involved… As the implementation starts, we can start planning the tests."*

It was also pointed out that requirements should be reviewed with all of the parties before the start of the implementation. It was also mentioned that unambiguous requirements that are easy to find help with the challenging testing schedules.

Interfaces

It was pointed out that documentation related to interfaces, and the import files should be very detailed. Documentation should include things such as: which data types and data fields can the interface contain, terms and conditions related to interfaces, how to handle error situations, and how to use and authenticate the interfaces. It was also mentioned that the interface documentation should be ready for customers to use.

Configurations

The desired information related to configurations seems to consist of the following things: configuration instructions, the status of configuration, how configurations work in different situations, and how to manage the configurations. It was also pointed out that configuration guides are important and that they should be more firmly connected to actual work process.

Modifications, exceptions, bugs and errors

Development team pointed out that exceptions and modifications should be documented. During the specification process, the exceptions should be documented to help developers during the upcoming development process. During the development process, the modifications and possible exceptions to the original plan should also be documented. Modifications and repairs should be documented, because these are important things to know for the support team. Modifications are important also from the point of view of customer end. It was also pointed out that typical errors and the operational models of the error situations should be documented.

Instructions

It was pointed out that clear operating instructions and installation guidelines are important to document. Configuration instructions are also needed. It was mentioned that the instructions should contain information about the steps that need to be taken:

- *"The instructions of how to put things to use are important. We need to know what needs to be done to reach certain end result. Also directions like 'if you do this, do also this' help."*

It was also mentioned that possible online helps would be useful. In addition to these, it was pointed out that customers need operating instructions.

Use cases

It was mentioned that use cases and use case scenarios should be documented. However, this wasn't specified more precisely. Development team pointed out that before the actual development, the use cases should be created in order to help developers. It was also pointed out that from the business perspective the use cases are desirable since they help to understand how the system actually works in real life.

Database description and architectural description

It was pointed out that the documentation should include description of database and its structure, and also the possible links to other databases. It was also mentioned that the

description of the structure of the database should be more detailed. About the architectural description, it was pointed out that some developers desire the description of the architecture to support their work, although this wasn't mentioned by any developers themselves.

4.4.3 How should documentation be produced?

To the question of how should the documentation be produced, there were several things mentioned and most of them were related to the documentation style, tools and format, or when to document and where to keep the documents. It was pointed out that there is a need for lightweight method to produce clear and good documentation in agile way. The latest versions of documents should be easy to find and it should be very clear which one is the latest version. All in all, it depends on the type of document which format, style or tools fit.

Documentation Style

About the documentation style it was mentioned that the descriptions of how the system works should be short and very clear. From the sales perspective, it was also pointed out that the documents should be short, and long textual descriptions and self-evident information should be left out. It was also pointed out in the interviews that pictures support the text and illustrate things better compared to long textual descriptions. When it comes to UML diagrams there were divided opinions, some participants pointed out that they don't like UML diagrams and others said that they find them useful when used in the right situations and when people know which diagram to use. Some comments related to the use of UML diagrams were:

- *"In the right situations UML diagrams are great, but people need to know which diagram to use in which situation. For example Sequence diagrams are useful in some situations."*
- *"I have a personal experience of a situation where there was a complicated diagram in the use, and it was really confusing and hard to keep up with, so I made kind of a summary of the main points with bullet points and it helped me significantly."*

- *"From the business point of view, use cases are good to describe how the solutions work in real life situations. Use cases are more useful than only descriptions of features."*

About technical documentation it was mentioned that examples of how to actually use the system helps a lot to understand the content, and given example related to this was integration descriptions. About acceptance criteria it was mentioned that it doesn't really matter how they are presented if they are at least presented at all and even the bullet points would be good enough.

In figure 11 the average of the answers to the survey question "*How useful you find each of the following documentation styles?"* are presented. The rating scale was chosen to be from one to four to prevent participants to choose the middle option. One means not useful, two means somewhat useful, three means useful, and four means very useful. From the figure 11 we can see that overall all of the chosen documentation styles are rated with somewhat same, except the pictures of mockups, wireframes or prototypes seem to have a higher rate. However, there were altogether 18 forms filled but two of the participants left the UML diagrams section blank, and one participant left the pictures of mockups / wireframes / prototypes section blank, and this affects the average of the results. In the survey form there were also an option to add other styles, and one participant added presentations and rated it useful, but it is not considered here because it was only one answer.

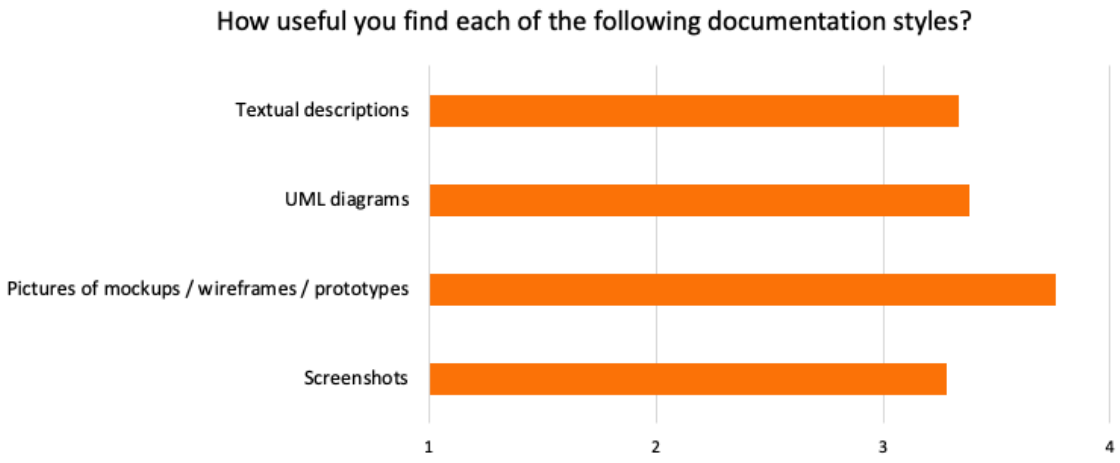

Figure 11. The average of participants' answers related to the questions of documentation style

Since some participants didn't fill all of the sections in this question in the survey, the blank sections are considered as the participant weren't familiar with the style or that they don't have an opinion about it since they don't use it.

Tools and Format

About the format, many of the participants pointed out that Word or PDF documents shouldn't be used, rather there should be some kind of online format in the use. However, it was also pointed out that if the documentation format is web-based there should still be an option to download a PDF file. As participants said that tools like Word shouldn't be used, they usually mentioned that some structural solution would be great. In addition to structural solution, the participants pointed out that the solution should be dynamic and there should be a possibility to make links and references between different pages. Many participants mentioned Wiki as a possibly suitable tool. Also, an option to print or download a single page was desired, and one participant referred to this issue as follows:

- *"There should be a feature to print single pages so it wouldn't be compulsory to print a 150 page long document for the customer and tell them that the information they seek can be found at page 80."*

Couple of tools were mentioned by the participants, some tools were said to be bad and some good. One tool that was mentioned in many interviews was Confluence, which is kind of a Wiki-based tool for workspace sharing that the company is currently using. Participants seemed to like Confluence as a tool especially because it also has an inner drawing tool:

- *"Confluence offers a possibility to draw diagrams. From the point of view of specification and production Confluence is suitable tool for drawing diagrams, and it is good enough because the purpose is to produce simple information."*
- *"Confluence has a drawing tool, which everyone can use so everyone is able to maintain and update the information also in the diagrams."*

Although some weaknesses related to the use of Confluence were pointed out too:

- *"Confluence is great tool, but different teams have different spaces there and it is hard to find information from other teams' spaces."*
- *"Confluence is an easy tool if you know what you are looking for."*
- *"Confluence's search function is poor. The search needs to be really precise and you must to know what you are searching for."*

About Confluence it was also pointed out that formatting and editing tools are not used enough. It was pointed out that it would make the pages much easier to read if the text were formulated:

- *"If the text is formatted it is easier to read, and if for example code samples are styled with different font than the rest of the text. The visual view is important and it also speeds up the reading process."*

One interviewer pointed out that many programming languages such as Python and Java have great documentation tools. It was also pointed out that Robot Framework has good user guides and descriptions of languages. From the point of view of user interface (UI) design, Adobe XD was pointed out as a possible future direction since it seems quite straightforward tool. Overall it was pointed out that web-based tool would be good since they don't require installing softwares. It was mentioned that with the tool it should be easy to search, modify and read the information. In addition to this it was pointed out that the number of tools that are being used should be minimum. However, some participant said that it actually doesn't matter if there are many tools in the use, and if different types of documents are located in different places.

In figure 12 the average of the answers to the survey question "*How useful you find each of the following documentation tools?"* are presented. The rating scale here is also from one to four to prevent the "easy" answers. One means not useful, two means somewhat useful, three means useful, and four means very useful. Wiki, Word, Jira, and VCS were the given options but there was also the option to add other tools and overall six out of 17 participants added

Confluence, which is a tool being used in the case company, so it is also taken into account here. Altogether there were 17 replies to this question, however four participants didn't fill the Wiki section, one didn't fill the VCS section, and like said before six participants added the Confluence tool.

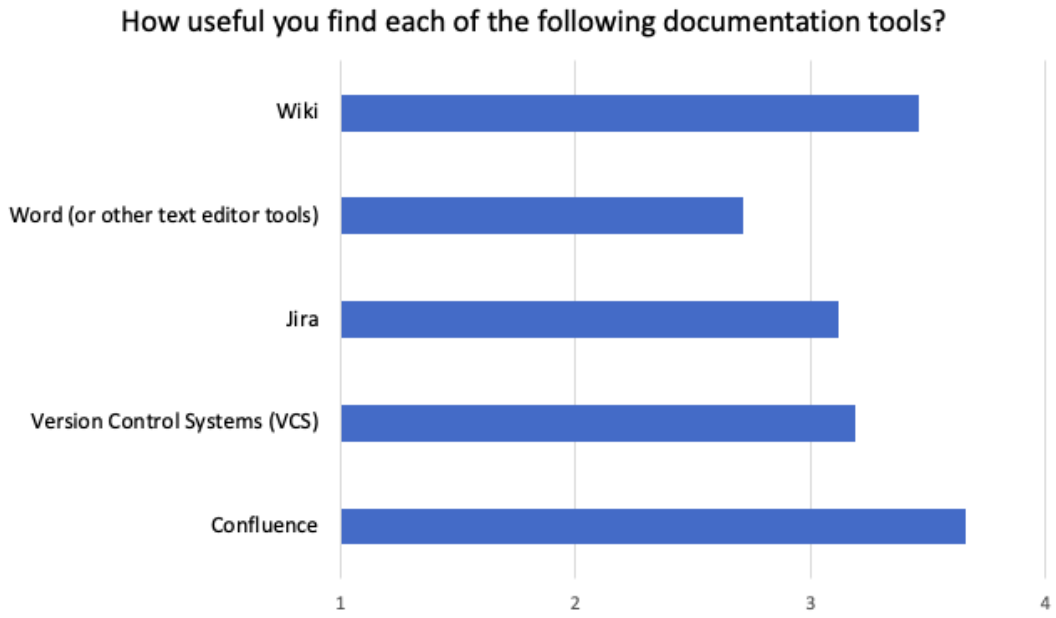

Figure 12. The average of participants' answers related to the questions of documentation tools

From the results presented in figure 12 it can be seen that apart from Confluence, Wiki was the highest rated tool. Jira and VCS are rated almost in the same way, and Word or other text editor tools were rated lowest of these chosen tools. Some other tools were also mentioned but since they were all mentioned only once they are not considered here. Other tools that were mentioned were PowerPoint, Adobe XD, Photoshop, Illustrator, and Therefore, which is also a tool that the company is using now. Again, since some participants didn't fill all of the sections in this question in the survey, the blank sections are considered as the participant weren't familiar with the tool or that they don't have an opinion about it because they don't use it.

When to document

All the participants who mentioned something related to the issue of when to document, were thinking that documentation should be done continuously during the development process. Comments related to this were:

- *"Documentation process should be integrated to the daily work as well as possible. Functionalities, plans, and testing should be a unified chain as a part of version control."*
- *"We should have simultaneous documentation with development, so things would be updated all the time."*
- *"Documentation should be a part of the chain. If it's not part of the chain then it's not up-to-date, it is randomly updated, and there are shortcomings and mistakes."*

It was also pointed out that documentation should be easy to maintain for the product development team, because otherwise it might not be the first task to be done. It was also mentioned that the documentation process should be more modular so to speak, so there wouldn't be so much work to do. Moreover, it was pointed out that no proper documentation should be done before the implementation, and that the amount of documentation produced in specification phase should be as little as possible. It was said that it's enough if user interfaces are drawn on paper, and the actual documentation is not produced until the implementation phase.

Where to keep the documents

About the issue of where the information can be found, it was pointed out that documentation should be in a place that is easy to find and access. It was also mentioned that documents should be easy to access also with the remote access. In addition to that it was said that documentation should always be available, and that all of the documentation should be located into one place. About the current situation, it was said that documents are now in different places and it is not clear which one is the newest version. Some of the comments

and opinions that the interviewees gave about the current situation and about how things should be are listed below:

- *"It is difficult because we have many different customer relationships, and every one of us (inside the company) puts the information in different places."*
- *"If only the one that is familiar with the document knows how to find it, it doesn't serve the purpose."*
- *"Currently documentation is hard to find, because it's in pieces here and there. Version control is bad. There should be a clear place where the documentation can be found."*
- *"It is important that the documentation is accessible and easy to find, otherwise it's not beneficial. And if there is useless documentation, time will also be wasted."*

It was also pointed out that it would be important to have a general view of what kind of documentation there is available and who has the responsibility to maintain each document. It was also mentioned that at least the internal documentation should be in the same place, and it would be great to have a one place for internal directives. However, it was also pointed out that maybe different types of contents should be in different places, or at least the audience of the document should be defined. Also, an idea of online portal, which would contain customer related documentation and which the customers could access by themselves was brought up.

4.4.4 Who or which team should produce the documentation?

The assumption regarding the answers of this question was that almost all of the interviewees will try to push the documentation work on someone else and probably to the product development team but surprisingly this wasn't actually the case. It was often mentioned that the development team should document the technical aspects, but it was also pointed out that everybody needs to document the things related to their work and that the key to success would be good communication and mutual understanding.

In figure 13 we can see the eight most common responses to the question of who should produce the documentation, which are: “everybody”, “everyone should document their own work”, “development team documents the technical things / the things related to the development”, “technical writer”, “someone should have overall responsibility”, “co-operation between different teams”, “marketing / sales / customer teams should help with the commercial aspects”, and “depends on document / target audience”. Most of the interviewees mentioned one or two of these things but some interviewees mentioned three. In one interview none of these things were mentioned.

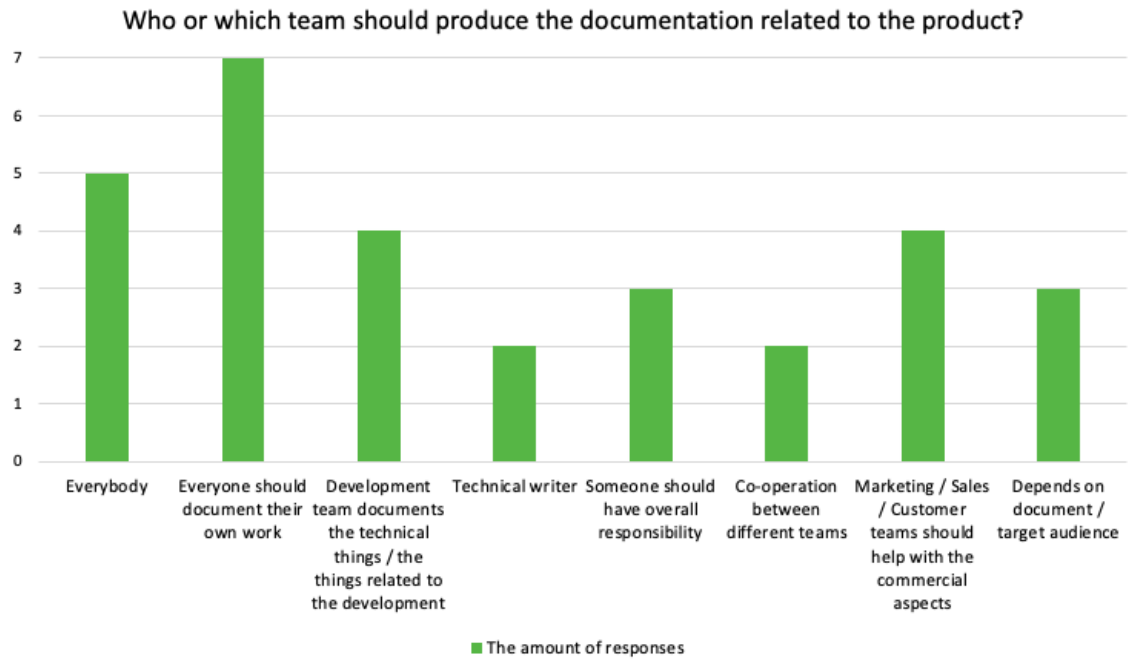

Figure 13. The most common answers related to the question of who should produce the documentation

As it can be seen from figure 13 the most common answer was that everyone should document their own work. The second most common answer was that everybody should document.

As said earlier, the assumption before the interviews was that the most common answer to this question would be that development team should produce the documentation. In four interviews out of fifteen it was mentioned that development team should indeed produce

documentation, but it was specified that they should produce the technical documentation or documentation related to the development. One comment related to this was:

- *"Developers should produce the documentation related to the product. If someone else produces, the information won't be correct. Developers should be in charge of technical details, and that others understand how things work."*

There were a lot of comments related to the answer "marketing / sales / customer teams should help with the commercial aspects", and "depends on document / target audience", but only one comment per interview was counted in to the results that are shown in figure 10. Some comments related these things were:

- *"Sales and marketing teams will help with visualization. Product management and product development have the knowledge."*
- *"In order to produce documentation that is understandable for customers, there needs to be someone, for example consult or someone that understand business aspects, aboard in the production so that the documentation won't be only technical."*

The answer "co-operation between different teams" was mentioned in that form only in two interviews, however the interviewees gave some other comments that are closely related to the co-operation and communication. Some comments related to this topic were:

- *"Co-operation! So everyone knows what others need."*
- *"Quite often there are requests that product development should produce documents for marketing team or some other team, but product development team don't know a lot about marketing aspects. In my opinion product development should produce material, for example functionalities and related documents, that can then be used to produce more commercial material. There should be co-operation!"*
- *"Product development team can produce for example documentation related to configurations, functionalities, or how to take something into action, but business related documentation should be produced by someone else."*

Overall the answers to this question varied, but a common theme can be found. It seems like all of the interviewees think that everyone has to do something so that the most useful documentation can be produced. Some roles were given more responsibilities than others but the answers were usually justified and it was explained why someone should produce certain type of documentation.

4.4.5 Other issues

In this subchapter the other things that were brought up in the interviews are presented. These things include documentation language, product versions, issues related to customers and partners, installation descriptions, and that different documentation types should be produced in different phases of the development process.

Language

One of the things that was brought up in many interviews was the issue with the documentation language. It was pointed out that now some of the documentation is in Finnish and some of the documentation is in English. Some of the documents are in both languages but many of the documents are only in one of these languages which means that they need to be translated if the customer requires the other language. One interviewee said:

- *"Do we even need documentation in Finnish?"*,

as the official language of the company is English, however, many interviewees pointed out that some or actually many of the customers require Finnish documentation. The issue here is that the documentation is needed in both languages but in which language the specific document is needed depends on the situation. One interviewee said:

- *"It would be nice if the documentation were in English, but then the text needs to be understandable, and if it's hard to write understandable text in English maybe it would be better to just write it in Finnish and someone will translate it"*.

About the documents that are written in Finnish, it was pointed out that they are not internationally usable or actually usable anywhere else than in Finland, and they in some scale prevent hiring people that don't understand Finnish well, for example if the directions and guidelines are only in Finnish. One desire that was brought up was that there would be an agreement of the language that will be used in all cases, was it either English or Finnish, so that all of the documentation would be in the same language, and then the required documents would be translated when needed. In one interview it was pointed out that English should be the agreed language that is used, however it was questioned if it's worthwhile compared to documenting things in the native language. One issue related to language that was also brought up was that when there is a need to take screenshots for guides and manuals with both language, it is painful because in some cases in order to get to a certain situations several operations need to be taken, and because of two languages this needs to be done two times.

Product versions

The difficulties related to the different versions of the product and how to handle the versions were mentioned in quite many interviews. First of all, it was pointed out that there are lot of different versions so managing them is quite difficult, and that is why version control and good documentation is needed. It was also pointed out that currently the latest version is difficult to find and that it is also difficult to know which one is indeed the latest version:

- *"It would be important that all of the documents were in one place that is easy to find. Now, for example the latest version is hard to find. It should be found easily."*
- *"People don't know which one is the newest version."*
- *"It should be agreed where the latest version is, so we can trust that certain document is the newest."*

It was also pointed out that the documentation process related to the product versions is too slow:

- *"Now the production of documentation is really slow. When we get the document, it's already almost a time when the next document should be ready."*
- *"Sometimes there is a situation that a new version of the product is soon to be launched and we don't have any material related to it. Faster documentation is needed."*

In addition to the comments above, it was also mentioned that when the new version of the product is being launched, the license-based manuals should be ready and updated. It was also pointed out that it would be useful to have some kind of general view of upcoming things that the new versions will include for example in a "road map" -style. Online format was mentioned to possibly be a good format for product version documentation. It was also pointed out that version control related to imports would be very important:

- *"The old versions should be available. The old model imports are needed because some customers still use the old models, and that is why information related to them must be available."*

Customers & Partners

It was pointed out that customer specified documentation is important and that changes in product documentation should be customized. It was also pointed out that customers feel that some of the released documents are too technical, so it was suggested that customer documentation should be written in more understandable manner.

It was also mentioned that there is a new way of doing things, which is through partners, who also need documentation:

- *"It's our product but the partner company manages the sales and initialization. In this case we are the technical support for the product. Although now also the partners need documentation, and that is why tight and more formal way of documenting is needed. Internal documentation doesn't have to be so strict, but external*

documentation should be very clear and explicit, and things should be explained more in detail and precisely."

Installation descriptions

It was pointed out that the installation instructions are quite ambiguous, and sometimes even the own staff can't manage the installation. It was also mentioned that updating the instructions is difficult when the new features come in. It was also pointed out that the installation descriptions coming from developers are not suitable for customers. One interviewee pointed out the main issue:

- *"The challenge is who should manage these instructions related to deployment."*

Different documents in different phases

Many of the interviewees mentioned that documentation should be produced continuously during the development process, however, some interviewees also pointed out that different types of documents should be produced in different development phases. Basically, there are three options of when to produce the documentation: before the development, during the development, and after the development. In the group interview, the developers pointed out that before the development, developers need documentation that tells them what to do and why. They also pointed out that some specifications such as use cases and exceptions are needed before the actual implementation. It was pointed out that during the development, things like changes and exceptions to the plans are needed. It was also mentioned that during the development phase, documentation should be created as code comments and for example as Javadocs, which support the code and explain things more closely. About the final documentation, it was pointed out that things like configurations, features and functionalities should be documented. It was also pointed out that profound documentation shouldn't be done before the development:

- *"No proper documentation before the implementation. The specification phase should be as little as possible. The final documentation should be produced together with the implementation."…"It's enough that user interfaces are drawn on paper."*

In addition to this, it was also pointed out that basically there should be different documents for specification phase and implementation phase, and that the final documentation shouldn't be the defining factor. It was also mentioned that during the development, the focus should be on the actual development and not on the final documentation. It was added that during the development the possibly produced documentation don't have to be in its final form.

4.5 Conclusions of the Interviews

Overall the interviews offered quite good understanding of the needs related to the documentation of the product. After the interviews, it was easy to see which are the most critical things that need to be documented and why. The interviews also provided new aspects which need to be taken into account in order to produce the desired documentation in an understandable and easy way. Nine main categories of why the documentation of the product is needed were formed according to the interviews. Concerning the issue of what should be documented, 13 most important things were found. Opinions of documentation styles, tools and format were given in the interviews and also the overall opinions related to the issue of how the documentation should be produced were discussed. The responses related to the issue of who should create the documentation were divided into eight groups by the interview results. Some other things that were brought up in the interviews are related to documentation language, product versions, customers and partners, installation, and to the issue that different documents should be produced in different development phases.

5 DISCUSSION AND THE PROPOSAL OF THE FUTURE DIRECTION OF CASE COMPANY'S PRODUCT DOCUMENTATION

This chapter aims to answer to the research questions of this thesis. The answers to the questions are presented in a form of proposition. The proposition contains information about what the documentation includes, how the information is presented, and when the documentation is created and by who. As a reminder, the research questions of the thesis are:

- What should be documented related to the product during its life cycle in agile software development process?
- How and when should the documentation related to the product be created?
    - Which tools should be used?
    - Which style of documenting should be used?
- Who should produce the documentation?

The answer to the first research question, which deals with the issue of what the documentation should include is mostly composed from the interview results, because the content of documentation is in most cases dependent on the product, team, and project, and therefore no specific answer can be found from the literature. The other two questions are formed from both the interview results and the knowledge gathered during the literature research. The most important things to document, according to both empirical study and the literature review, seem to be requirements, features, functionalities, configurations, interfaces, use cases, test cases, design and architecture, and instructions and help guides. Also marketing documents, and documentation for customers and partners are needed, however these are not introduced in this chapter since they won't directly affect the product.

About the tools that should be used, anything else is not suggested except that a wiki-based tool seems to be the best fit judging by both the empirical study and the literature review. Wiki is a great tool because it allows its users to make structures and that way the information, which in this case means the documentation, can be divided into smaller pieces. Wiki also allows its users to make links between different pages, which again makes it

possible to distribute the information. These features make the documentation much easier to read and at the same time speeds up the reading process, however in order to achieve the better and faster readability, the information must be well structured. The case company is currently using a tool named Confluence, which is a wiki-based tool, however as pointed in the interviews the search of information is quite hard at the moment. The search function that Confluence offers is not very good, and especially if the reader doesn't know exactly what he or she is looking for it is quite difficult to find the certain information since the search string needs to be very specific. Confluence can be a great tool but if it's the tool to be used, perhaps some re-structuring of the information is needed.

After all, the decision of what to document, the amount of information, how to present the information, where to place it, and which tools to use, are by the team to decide. Here are some guidelines based on the literature research to help with these decisions:

1. Document only the critical information and leave out the obvious information.
2. Single source information. Try to document information only once, in the best place possible, whether it is in the source code or in external documents.
3. Try to combine different documents.
4. Use reusable documents if it's possible.
5. Specify the target audience before documenting and use different styles and techniques to make it as beneficial as possible for the target audience.
6. Consider if the information is already stabilized. If it's not, try to document it as easily and shortly as possible in order to prevent redocumenting.

5.1 Use Cases and User Stories

UML diagrams were discovered to be a good documentation technique during both the literature research and the empirical study. Use cases and user stories seem to be the best techniques to document requirements. Documenting use cases can be useful for various stakeholders, and because use cases can be used to describe the business aspects or the technical aspects of the system, they are useful for example when creating customer documentation or for developers as they work. It was mentioned by many interviewees that use cases should be documented. The same was noticed during the literature review, however

the most recent studies seem to point out that user stories are the way to document requirements. Although it should be noted that use cases or user stories alone are not enough as documents but they are a really good addition to requirements documents, and if wiki is used as a tool, use cases can be linked to pages that contain information about requirements specification and vice versa, and in that way the reader can explore the certain topic more if it's needed.

5.2 Requirements Specification

Documenting requirements, and acceptance criteria related to requirements were pointed important by the interviewees. The importance of documenting requirements was also showing during the literature research. However, there seem to be few problems related to documenting requirements during agile development processes. One problem is the possibility of changing requirements, and another problem is ambiguous requirements and the uncertainty about the status of the requirement. Luckily both the empirical study and the literature review offered a possible solution to the problems related to requirements. The possible solution to these problems would be to add some regulations. The minimum accuracy that the description of requirement must fulfill should be defined. Also determining the "definition of ready", which defines the readiness of the requirement, would help with the approval of the features. In addition to these, adding acceptance criteria for the requirements (or at least for the most critical requirements if it's not possibly to do this for all the requirements) would help significantly to decide whether the requirement can be stated accomplished or not.

Documenting features, functionalities, configurations, and different interfaces are also desired by the interviewees. These elements are closely related to each other and also closely related to requirements, and that is why it would be wise to make links between them and make their documentation process in a way part of the requirements specification. Features and functionalities are usually described by using requirements, which on the other hand are usually presented as use cases and user stories.

5.3 Test Case Specification

It was pointed out during the interviews that developers need to know what have been tested by the software QA team. Test cases are important to document since they help developers to see for example which functional requirements have been tested and if they passed or failed. Test cases are usually documented with textual descriptions, however when agile methods are used, the list with bullet points is enough as a documentation technique. Test cases are usually related to the user stories, which describe the requirements and that is why with using wiki as a tool, it would be possible to create a link between the test case and user story (or use case).

5.4 Design and Architecture

Documenting architectural design is important in order to understand how the system works and how different parts are linked to each other. During the empirical study it was observed that one of the main reasons why the interviewees seem to need documentation is to understand the system, its different solutions, and the links between them. Also during the literature research, system architecture was observed to be important issue to document. It was also noticed that in addition to documenting the architectural design, the architecture decisions seem to be extremely important to document in order to understand the key aspects of the architecture in the future. This understanding is needed by someone who needs to understand the decisions behind the architecture like for example developer or another architect. The importance of documenting the structure of database was also brought up in the interviews. Some aspects related databases, such as instructions, are already documented by the case company, but interviewees desire more detailed documentation related to the database structure. The structure of the database also communicates information about the system and its relationships, and that is why it is included in the proposition of architectural design document.

5.5 Instructions and Help Guides

It was found out that interviewees desire better instructions related to for example configurations and installation. It should be noted that even when agile methods are used the instructions and help guides are something that is inevitable to document. If instructions are not documented, it causes a huge amount of workload for someone who needs to answer the questions related to them. Also, if documentation is done using reusable documents, making changes and updates shouldn't be too time consuming. In addition to installation and configuration guides, user guides should also be produced. User guides can contain information about features and functionalities and how to use the system, or at least how to use the main functions. Among the other information in the user guide, it would be good to specify the target audience in the beginning. Also, the possible error situations and how to handle them would be good to document. If there is enough resources, an online version of the user guide would be useful, and the maintenance process would also probably be easier.

5.6 Proposition

In this subchapter the actual proposition is presented. First the hypothetical structure of wiki is presented. The structure of the wiki answers to the first research question, which concentrates on what should be documented during the agile software development process. After that, table 3 offers answers to the two other research questions. The other two questions concentrate on the issues of who should produce the documentation, and how and when it should be done. At the end of this subchapter figure 14 represents the overview of the produced documentation artifacts during the development process. In other words, figure 14 visualizes the big picture of the development flow and places the documentation artifacts presented in table 3 into the timeline. However, it should be noted that the timeline doesn't mean that the documentation artifacts should be only produced in that certain phase, rather it's a directional example.

As mentioned before, the most important things to document were found out to be requirements, features, functionalities, configurations, interfaces, use cases, test cases, description of design and architecture, and instructions and help guides such as configuration

guide, installation guide, and user guide. The structure of wiki, which includes all of these things could be for example following:

1. **Requirements Specification**
   - ***1.1. Features***
     - Description of the feature
     - How does the feature work
     - Acceptance criteria (for the most critical features)
     - Definition of ready (that describes which are the criteria that must be fulfilled before the feature can be approved)
     - Configurations → Can contain only a link to 1.4.
     - Functional requirements → Can contain only a link to 1.2.
   - ***1.2. Functional requirements***
     - Requirement status (draft / approved / reviewed) for each requirement
     - Each requirement includes information related to the following things:
       - Author / owner / who is responsible
       - What
       - Why
   - ***1.3. Non-functional requirements***
     - Requirement status (draft / approved / reviewed) for each requirement
     - Each requirement includes information related to the following things:
       - Author / owner / who is responsible
       - What
       - Why
   - ***1.4. Configurations***
     - The status of configuration
     - Configuration descriptions
     - Configuration instructions
       - How the configurations work in different situations
       - How to manage the configurations
   - ***1.5. Interfaces***
     - Different types of interfaces and their descriptions
     - Information related to:
       - Data fields and data field terms
       - Which data types the interface can include
       - Interface usage
       - Interface authentication and how it can be done
       - How to handle error situations
       - Data logging

1.6. Use Cases

1.6.1. Business Use Cases

1.6.2. Technical Use Cases

2. Functionalities

- Description of functionality
- How the functionality can be deployed
- How the functionality can be linked to other systems
- Changes and updates in functionalities:
  - The information about how the functionality has changed and how it is supposed to work now
  - Information that the functionality has been updated
- Configurations → Can contain only a link to 1.4.

3. Test Case Specification

4. Design and Architecture

4.1. Architectural Design

- Architecture decisions
  - Key structural elements
  - Externally visible properties of the key elements
  - Relationships between elements
  - Rationale of decision
- Key design principles
- Architectural specification
  - High-level description of the structure

4.2. Database

- Description of database design
- Description of database schema
- Data relationships

5. Instructions and Help Guides

5.1. Installation

5.2. Configuration guide

5.3. User guide

The proposed things to document are listed in table 3. The issues of how to represent the information, when to capture the information, and who should create the documentation are also presented in table 3. In addition to this each documentation artifact has a level of detail in which it should be documented and the target audience whom the documentation is directed.

Table 3. Proposed documentation artifacts and how they should be produced

Documentation artifact	Level of detail	How to represent	When to document	Who creates	Target audience
Functional requirements	High-level	User stories, Acceptance criteria	Pre-development (although because of possible changes in requirements, needs to be updated during the development)	Requirements are defined together with the customer → Developers and Software quality assurance are involved in the writing process User stories can be written by any team member	Software quality assurance, Software development, Product development
Non-functional requirements	High-level	User stories, Acceptance criteria	Pre-development (although because of possible changes in requirements, needs to be updated during the development)	Requirements are defined together with the customer → Developers and Software quality assurance are involved in the writing process User stories can be written by any team member	Software quality assurance, Software development, Product development
Features	Detailed	Textual descriptions, Functional requirements (Use cases can be used)	Post-development	Developers	Customer consultation service, Partner manager, Project services, Software development, Product development

Configurations	Detailed	Textual descriptions	Defined pre-development, but the actual documentation can be done post-development	Developers	Customer consultation service, Partner manager, Project services, Software development, Product development
Interfaces	Detailed	Textual description	Post-development	Software solutions	Customer consultation service, Service operations
Functionalities	Detailed	Textual description, Use cases	Post-development	Developers	Customer consultation service, Partner manager, Project services, Software development, Product development
Use cases	High-level	Use case scenarios	Pre-development	For example business analyst (although this task shouldn't be specified to a certain job title, rather a person who understands the business value and has also technical understanding would be the best fit to write use cases)	Software development, Product development
Software architecture	High-level	UML diagrams	Pre-development	Architect	Software development, Product development

Architectural decisions	High-level	User stories, Usage scenarios	Pre-development	Architect	Software development, Product development
Database architecture	Detailed	Entity-relationship (ER) diagrams	Pre-development	Architect	Service operations, Product development, Software development,
Source code and comments	Low-level	Code comments to clarify and add important information, Meaningful variable names	During development	Developers	Software development, Product development
Test case specifications	Detailed	Test cases	During development	Software quality assurance	Software quality assurance, Software development, Product development
Installation guide	Detailed	Textual (step by step) description, Pictures (for example screenshots) to visualize	During development / Post-development	Software solutions, Technical writer	No specified target group
Configuration guide	Detailed	Textual (step by step) description, Pictures (for example screenshots) to visualize	During development / Post-development	Developers, Technical writer	Customer consultation service
User guide	Detailed	Textual description, Pictures (for example screenshots) to visualize	During development / Post-development	Technical writer	Users of the system

Table 3 provides information related to each artifact, but in order to see the bigger picture, these documentation artifacts are also shown in figure 14. In figure 14 the development flow of documentation is visualized, and the documentation artifacts are placed in the order.

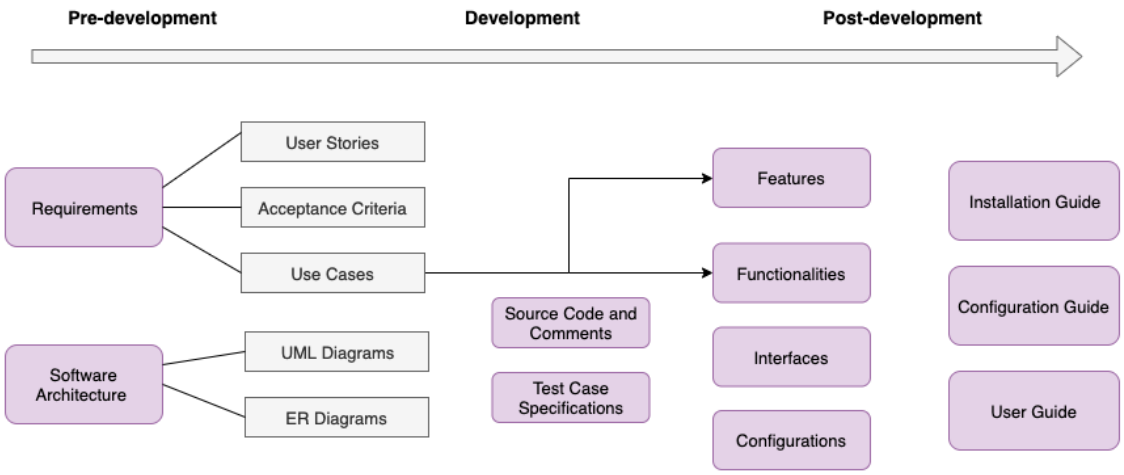

Figure 14. Overview of the produced documentation artifacts during the development process

The documentation artifacts presented in figure 14 are placed in the timeline based on the fact where they should be ready and documented. This means that even though some documentation artifact is placed on post-development phase, some information related to it might be needed in earlier phases too, even though the actual and final documentation shouldn't be done before this phase. For example features are suggested to be documented very late, but some stakeholders might require information related to them sooner. On the other hand, when using agile methods, the daily meetings are opportunities to discuss things face-to-face, and this can reduce the amount of documentation that is needed. The reason why many of the documentation artifacts would be good to document as late as possible is that the possibility that the information has stabilized and won't change anymore is bigger.

If we look at the timeline in the figure 14, it can be seen that requirements and software architecture should be documented early in the development process. This means that before the actual implementation, requirements should be specified and acceptance criteria related to them should be defined. However, also the features and functionalities need to be discussed before the implementation, but extensive documentation related to them shouldn't be done early, because there might be inevitable changes coming from customers during the

development. As requirements are specified before the actual implementation, also the user stories and use cases related to them should be created in this phase. User stories and use cases on the other hand communicate information about features and functionalities.

In addition to documenting requirements, the software architecture should be documented before the implementation. Software architecture here means the actual description of the software architecture captured for example with UML diagrams, and also the architectural decisions and database architecture. The instructions and help guides are placed in post-development phase, although there will probably be a need to at least partly document them during the development. However, installation guide, configuration guide, and user guide are placed in the post-development phase in figure 14 since it's not agile way to document information when the possibility that the information might change is high.

6 CONCLUSIONS AND SUMMARY

Even though the issue of how much documentation should be produced during the agile software development processes has been studied for several years, there are still no clear and unambiguous findings. The first publications (which are included in this thesis) related to the topic are published already in 2001. About the amount of documentation in agile projects it is often referred that "just enough" documentation should be produced, not more, not less. However, the phrase "just enough" is very nonspecific, and overall it's pointed out that in the end it is up to the project team to decide about the documentation. However, some general guidelines and good practices have been discovered. For example, only the critical information should be documented during agile projects. Combining documents and using reusable documents is also recommended. Overlapping should be avoided by documenting information only once in the best place possible. The specification of the target audience helps to choose the most convenient documentation techniques to serve the purpose. One thing that should be consider is when to document, since for example redocumenting can possible be avoided if documentation is done when the information has stabilized. Also, an agile document should capture the critical information without wasting time making it look nice, and that means that it might be wise that there would be someone who is responsible for the visual look of the documentation overall.

The aim of this thesis was to form a proposition of how the case company could start producing documentation related to their product in the future. The case company wanted to know what are the most critical things to document, how they should be documented and when should the documentation be created. Since the case company is using the Scaled Agile framework, which is a framework that combines lean and agile methods, the scope of this thesis was narrowed to the product documentation in agile environment. The research was conducted with literature review and empirical study. Snowballing was chosen as research method for the literature review part and the empirical part was executed by interviewing the product stakeholders from the case company. The proposition was formed by combining the results from the interviews and the knowledge gathered during the literature review. Requirements, features, functionalities, configurations, interfaces, use cases, test cases,

description of design and architecture, and instructions related to configurations and installation were found to be the most critical things to document.

REFERENCES

Agilealliance.org, (2019). *Extreme Programming*. [online] Available at: https://www.agilealliance.org/glossary/xp/#q=~

Agilealliance.org, (2019). *Kanban*. [online] Available at: https://www.agilealliance.org/glossary/kanban/#q=~

Agilealliance.org, (2019). *Scrum*. [online] Available at: https://www.agilealliance.org/glossary/scrum/#q=~

Altexsoft.com, (2019). *Technical Documentation in Software Development: Types, Best Practices, and Tools*. [online] Available at: https://www.altexsoft.com/blog/business/technical-documentation-in-software-development-types-best-practices-and-tools/

Ambler, S. (2002). *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*. New York: John Wiley & Sons, Inc., p.

Ambler, S. (2005). *Agile/Lean Documentation: Strategies for Agile Software Development*. [online] Agile Modeling. Available at: http://agilemodeling.com/essays/agileDocumentation.htm

Ambler, S. (2005). *Single Source Information: An Agile Core Practice for Effective Documentation*. [online] Agile Modeling. Available at: http://agilemodeling.com/essays/singleSourceInformation.htm

Ambler, S. (2016) *When Should We Create a Document on an Agile Team?* [online] Disciplined Agile. Available at: https://disciplinedagiledelivery.com/agile-documention-strategy/ [Accessed: 23.5.2019]

Beck, K. (2000). *Extreme Programming Explained: Embrace Change*. Addison-Wesley.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J. and Thomas, D. (2001). Agile manifesto. [online] Available at: http://www.agilemanifesto.org

Briand, L. C. (2003). Software documentation: how much is enough?. In: *Seventh European Conference on Software Maintenance and Reengineering, 2003. Proceedings*. [online] Benevento: IEEE. Available at: https://ieeexplore.ieee.org/document/1192406

Cohn, M. (2019). *User Stories*. [online] Mountain Goat Software. Available at: https://www.mountaingoatsoftware.com/agile/user-stories

De Lucia, A. and Qusef, A. (2010). Requirements Engineering in Agile Software Development. In: *Journal of Emerging Technologies in Web Intelligence*. [online] ResearchGate, Volume 2 (3), pp. 212-220. Available at: https://www.researchgate.net/publication/228988887

De Souza, S. C. B., Anquetil, N. and de Oliveira, K. M. (2005). 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*. [online] Coventry: ACM, pp. 68-75. Available at: https://dl.acm.org/citation.cfm?id=1085331

Ersoy, I. B. and Mahdy, A. M. (2015). Agile Knowledge Sharing. In: *International Journal of Software Engineering (IJSE)*. [online] CSC Journals, Volume 6 (1). Available at: https://www.cscjournals.org/library/manuscriptinfo.php?mc=IJSE-152

Feduniak, O. (2016). *Agile documentation: best practices*. [online] Eastern Peak. Available at: https://easternpeak.com/blog/agile-documentation/?utm_source=medium&utm_medium=post&utm_campaign=agile-documentation

Fowler, M. and Highsmith, J. (2001). The Agile Manifesto. *Software Development Magazine*.

Forward, A. (2002). *Software Documentation - Building and Maintaining Artefacts of Communication*. Ottawa-Carleton Institute for Computer Science, University of Ottawa, Canada.

Forward, A. and Lethbridge, T. C. (2002). The relevance of software documentation, tools and technologies: a survey. In: *Proceedings of the 2002 ACM symposium on Document engineering*. [online] McLean, Virginia: ACM, pp. 26-33. Available at: https://dl.acm.org/citation.cfm?id=585065

Garousi, G., Garousi, V., Moussavi, M., Ruhe, G. and Smith, B. (2013). Evaluating Usage and Quality of Technical Software Documentation: An Empirical Study. In: *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*. [online] Porto de Galinhas: ACM, pp. 24-35. Available at: https://dl.acm.org/citation.cfm?id=2461003

Gastegger, M. and Zünd, S. (2015). Maintenance of technical and user documentation. [online] Available at: https://www.researchgate.net/publication/303883087_Maintenance_of_technical_and_user_documentation

Goetz, R. (2002). How Agile Processes Can Help in Time-Constrained Requirements Engineering. In: *International Workshop on Time-Constrained Requirements Engineering*.

Hakkarainen, J., Hartimo, M. and Virta, J. (2014). *Muisti*. Tampere: Acta Philosophica Tamperensia vol. 6, pp. 18.

Hanssen, G. K., Stålhane, T. and Myklebust, T. (2018). *SafeScrum® – Agile Development of Safety-Critical Software*. Trondheim: Springer, p. 195.

Hess, A., Diebold, P. and Seyff, N. (2017). Towards Requirements Communication and Documentation Guidelines for Agile Teams. In: *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*. [online] Lisbon: IEEE, pp. 415-418. Available at: https://ieeexplore.ieee.org/document/8054887

Highsmith, J. and Cockburn, A. (2001). Agile Software Development: The Business of Innovation. *Computer*, [online] Volume 34 (9), pp. 120-127. Available at: https://ieeexplore.ieee.org/document/947100 [Accessed 13.5.2019]

Hoda, R., Noble, J. and Marshall, S. (2012). Documentation strategies on agile software development projects. In: *International Journal of Agile and Extreme Software Development*. [online] Inderscience Enterprises Ltd., Volume 1 (1), pp. 23–37. Available at: https://www.inderscience.com/info/inarticle.php?artid=48308

Hotomski, S., Charrada, E. B. and Glinz, M. (2016). An Exploratory Study on Handling Requirements and Acceptance Test Documentation in Industry. In: *2016 IEEE 24th International Requirements Engineering Conference (RE)*. [online] Beijing: IEEE, pp. 116-125. Available at: https://ieeexplore.ieee.org/document/7765517

Ibanez, L. (2018). *Agile Development: User stories are the new requirements document*. [online] Medium. Available at: https://medium.com/@LazaroIbanez/agile-development-user-stories-are-the-new-requirements-document-c105947c9291

Leffingwell, D. (2018). *Welcome to Scaled Agile Framework® 4.6!* [online] Scaled Agile. Available at: https://www.scaledagileframework.com/about/

Lehtonen, T., Tuomivaara, S., Rantala, V., Känsälä, M., Mäkilä, T., Jokela, T., Könnölä, K., Kaisti, M., Suomi, S., Isomäki, M. and Ylitolva, M. (2014). *Sulautettujen Järjestelmien Ketterä Käsikirja*. Turku: Painosalama Oy.

Lethbridge, T. C., Singer, J. and Forward, A. (2003). How software engineers use documentation: The state of the practice. In: *IEEE Software*. [online] IEEE, Volume 20 (6), pp. 35-39. Available at: https://ieeexplore.ieee.org/document/1241364

Maciaszek, L., A. (2007). *Requirements Analysis and System Design*. 3rd ed. Harlow: Addison-Wesley.

Mall, R. (2018). *Fundamentals of Software Engineering*, 5th ed. Delhi: PHI Learning Private Limited.

Mehrabian, A. (2013). The Scaled Agile Framework: Foundations of the Scaled Agile Framework® (SAFe). [online] Available at: https://www.sdjug.org/docs/SAFeForTechnologyLeaders-SDJug.pdf

Paetsch, F., Eberlein, A. and Maurer, F. (2003). Requirements Engineering and Agile Software Development. In: *Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*. [online] Linz: IEEE, pp. 308-313. Available at: https://ieeexplore.ieee.org/document/1231428

Pfahl, D. (2014). Lecture 11: Agile/Lean Methods. [online] Available at: https://courses.cs.ut.ee/MTAT.03.094/2015_fall/uploads/Main/SE2014-handout11.pdf

Radigan, D. (2019). *Kanban*. [online] ATLASSIAN Agile Coach. Available at: https://fi.atlassian.com/agile/kanban

Rehman, F., Maqbool, B., Riaz, M. Q., Qamar, U. and Abbas, M. (2018). Scrum Software Maintenance Model: Efficient Software Maintenance in Agile Methodology. In: *2018 21st Saudi Computer Society National Computer Conference (NCC)*. [online] Riyadh: IEEE, pp. 1-5. Available at: https://ieeexplore.ieee.org/document/8593152

Rico, D. F., Sayani, H. H. and Sone, S. (2009). *The Business Value of Agile Software Methods: Maximizing ROI with Just-in-time Processes and Documentation.* J. Ross Publishing.

Robillard, M. P., Marcus, A., Treude, C., Bavota, G., Chapparo, O., Ernst, N., Gerosa, M., A., Godfrey, M., Lanza, M., Linares-Vásquez, M., Murphy, G. C., Moreno, L., Shepherd, D. and Wong, E. (2017). On-Demand Developer Documentation. In: *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. [online] Shanghai: IEEE. Available at: https://ieeexplore.ieee.org/document/8094446

Rubin, E. and Rubin, H. (2010). Supporting Agile Software Development Through Active Documentation. In: *Requirements Engineering*. Volume 16 (2), pp.117.132. Available at: https://link.springer.com/article/10.1007/s00766-010-0113-9

Scaled Agile Inc., (2018). *Agile Teams*. [online] SAFe. Available at: https://www.scaledagileframework.com/agile-teams/

Scaled Agile Inc., (2018). SAFe® 4.6 Introduction: Overview of the Scaled Agile Framework® for Lean Enterprises. [online] Available at: https://www.scaledagile.com/resources/safe-whitepaper/

Scaled Agile Inc., (2018). SAFe – The World's Leading Framework for Enterprise Agility. [online] Available at: https://issuu.com/scaledagile/docs/agility-brochure-october-2018-final

Scaled Agile Inc., (2019). *About Us*. [online] SAFe. Available at: https://www.scaledagile.com/about/about-us/

Scott, S. (2016). *4 Steps to Providing Useful Documentation in an Agile Environment*. [online] SmartFile. Available at: https://www.smartfile.com/blog/4-steps-providing-useful-documentation-agile-environment/

Selic, B. (2009). Agile Documentation, Anyone? In: *IEEE Software*. [online] IEEE, Volume 26 (6), pp. 11-12. Available at: https://ieeexplore.ieee.org/document/5287001 [Accessed 11.5.2019]

Sillitti, A., Succi, G. (2005). Requirements Engineering for Agile methods. In: Aurum, A. and Wohlin, C. ed., *Engineering and Managing Software Requirements*. Berlin: Springer, pp. 309-326.

Sommerville, I. (2001). Software Documentation. In: *Software Engineering*, 4th ed. Boston: Addison-Wesley.

Sommerville, I. (2011). *Software Engineering*. 9th ed. Boston: Addison-Wesley.

Steinberger, J. and Prakash, A. (2011). ModelDoc: Auto-generated, Auto-regenerated Wiki-Based Database Documentation. [online] SemanticScholar. Available at: https://www.semanticscholar.org/paper/ModelDoc-%3A-Auto-generated-%2C-Auto-regenerated-Steinberger-Prakash/27986955ab59ffe28108791c1d8655c9bdc4d260#extracted

Stettina, C. J. and Heijstek, W. (2011). 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*. [online] Pisa: ACM, pp. 159-166. Available at: https://dl.acm.org/citation.cfm?id=2038509

Stettina, C. J. and Kroon, E. (2013). Is there an Agile Handover? An Empirical Study of Documentation and Project Handover Practices Across Agile Software Teams. In: *2013 International Conference on Engineering, Technology and Innovation (ICE) & IEEE International Technology Management Conference*. [online] The Hague: IEEE, pp. 1-12. Available at: https://ieeexplore.ieee.org/document/7352703

Turk, D., France, R. and Rumpe, B. (2002). Limitations of Agile Software Processes. In: *Third International Conference on Extreme Programming and Flexible Processes in Software Engineering*. [online] Alghero: Arxiv, pp. 43-46. Available at: https://arxiv.org/abs/1409.6600

Turk, D., France, R. and Rumpe, B. (2005). Assumptions Underlying Agile Software Development Processes. In: *Journal of Database Management*, Volume 16 (4), pp. 62-87. [online] Available at: https://arxiv.org/pdf/1409.6610.pdf

Tyree, J. and Akerman, A. (2005). Architecture Decisions: Demystifying Architecture. In: *IEEE Software*. [online] IEEE, Volume 22 (2), pp. 19-27. Available at: https://ieeexplore.ieee.org/document/1407822 [Accessed 16.5.2019]

VanAlbrecht, A. and Nemani, R. (2014). Documenting Agile Project Knowledge: A Review of Knowledge Capture for Agile Practices. In: *International Journal of Computer Science Engineering and Technology (IJCSET)*. [online] IJCSET, Volume 4 (6), pp. 194-199. Available at: http://ijcset.net/docs/Volumes/volume4issue6/ijcset2014040602.pdf

Voigt, S., Hüttemann, D. and Gohr, A. (2016). SprintDoc: Concept for an Agile Documentation Tool. In: *2016 11th Iberian Conference on Information Systems and Technologies (CISTI)*. [online] Las Palmas: IEEE, pp. 1-6. Available at: https://ieeexplore.ieee.org/document/7521550

Wagenaar, G., Overbeek, S., Lucassen, G., Brinkkemper, S. and Schneider, K. (2018). Working software over comprehensive documentation – Rationales of agile teams for artefacts usage. In: *Journal of Software Engineering Research and Development*. [online] SpringerOpen. Available at: https://doi.org/10.1186/s40411-018-0051-7

Wells, D. (1999). Extreme Programming: A gentle introduction. [online] Available at: http://www.extremeprogramming.org/

Wohlin, C. (2014). Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering. In: *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*. [online] London: ACM, Article No. 38. Available at: https://dl.acm.org/citation.cfm?doid=2601248.2601268

APPENDIX 1. PRODUCT DOCUMENTATION SURVEY FORM

Product Documentation Survey

Name: _____ Date: _____

1. How long have you been working on software industry?

   ☐ < 1 year
   ☐ 1-5 years
   ☐ 6-10 years
   ☐ more than 10 years

2. Which types of documents already exist or are currently produced related to the product?

   ☐ Requirements documents
   ☐ Design documents
   ☐ Code comments
   ☐ Test documents
   ☐ Help guides / Manuals
   ☐ Other documents: _____

3. What is the style of documenting that is being used in the documentation related to the product?

   ☐ Text format
   ☐ UML diagrams
   ☐ Pictures (Screenshots / Mockups / Wireframes / Prototypes / Other)
   ☐ Other: _____

4. Which tools are used to document the product?

   ☐ Wiki
   ☐ Word (or other text editor tools)
   ☐ Jira
   ☐ Version Control Systems (VCS)
   ☐ Other tools: _____

5. How important do you consider documentation related to the product? (1 = Not at all important, 2 = Slightly important, 3 = Somewhat important, 4 = Extremely important)

   1 2 3 4

6. How useful you find each of the following documentation styles? (1 = Not useful, 2 = Somewhat useful, 3 = Useful, 4 = Very useful)

	1	2	3	4
Textual descriptions	☐	☐	☐	☐
UML diagrams	☐	☐	☐	☐
Pictures of mockups / wireframes / prototypes	☐	☐	☐	☐
Screenshots	☐	☐	☐	☐
Other: _____	☐	☐	☐	☐

7. How useful you find each of the following documentation tools? (1 = Not useful, 2 = Somewhat useful, 3 = Useful, 4 = Very useful)

	1	2	3	4
Wiki	☐	☐	☐	☐
Word (or other text editor tools)	☐	☐	☐	☐
Jira	☐	☐	☐	☐
Version Control Systems (VCS)	☐	☐	☐	☐
Other tools: _____	☐	☐	☐	☐

APPENDIX 2. INTERVIEW QUESTIONS

Interview - Product documentation

1. Describe your team. What is the size of your team? Is the whole team physically co-located?

2. What is your role in the team?

3. Describe the product briefly. How are you involved with the product?

4. Why do you need the documentation of the product?

5. What should be documented related to the product?

6. How should the documentation be produced? Which styles of documentation should be used? Which tools should be used?

7. Who or which team should produce the documentation related to the product? Why?