TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Karl-Ivar Pajula 142423 IABB

PROCESS, BENEFITS, AND COST OF DOCUMENTATION AMONG ANALYSTS IN AGILE SOFTWARE DEVELOPMENT TEAMS

Bachelor's thesis

Supervisor: Alexander Horst Norta
PhD

Tallinn 2018

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Karl-Ivar Pajula 142423 IABB

DOKUMENTEERIMISE PROTSESS, KASULIKKUS JA KULU ANALÜÜTIKUTE SEAS AGIILSETES TARKVARAARENDUSE MEESKONDADES

Bakalaureusetöö

Juhendaja: Alexander Horst Norta
PhD

Tallinn 2018

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Karl-Ivar Pajula

21.05.2018

Abstract

Today, software development teams are using Agile methodologies, which emphasize communication with the client, flexibility, adapting to the situation and planning the development in one to two-week cycles. This model is based on the Agile Manifesto, which was published at the beginning of 21st century [1]. In addition, Agile concept recommends minimal documentation of solutions. At the same time, there are existing theories and publications, which claim the opposite and describe different challenges if teams are trying to combine several methods. The purpose of this thesis is to investigate one of the challenges – documentation, its process, benefit and cost among analysts. In order to solve the problem, the author conducts interviews with analysts and analyzes the results qualitatively. Based on the results, the author presents the generic process of documentation which can be followed and to depend on if there are questions about the format and details of documentation. Also, it turns out that the main reason for documenting at all is to aid the development and it takes about a third of one week. Reading the documentation takes less time but the thesis describes situations how documentation which is as short as possible and as detailed as necessary can save the time for the whole team. In conclusion, every team must agree within themselves and also with the client how much and what is documented in the project. At the end of the thesis, the author brings out some recommendations to follow.

This thesis is written in English (United States) and is 50 pages long, including 7 chapters, 7 figures and 4 tables.

Annotatsioon

Dokumenteerimise protsess, kasulikkus ja kulu analüütikute seas agiilsetes tarkvaraarenduse meeskondades

Täna, tarkvaraarenduse meeskondades on peamiselt kasutusel agiilsed metoodikad, mis rõhutavad suhtlust kliendiga, paindlikust, kohandumist olukorraga ning arenduse planeerimist ühe kuni kahenädalaste tsüklite kaupa. Antud mudelile pandi alus 21. sajandi algusaastatel Agiilse Manifestoga [1]. Lisaks eelnevalt mainitutele, soovitab agiilne kontseptsioon minimaalset lahenduste dokumenteerimist. Samal ajal eksisteerivad teooriad ning publikatsioonid, mis räägivad vastupidist ning kirjeldavad erinevaid väljakutseid, kui meeskonnad üritavad kombineerida mitut alternatiivi. Käesoleva lõputöö eesmärk on uurida ühte väljakutsetest – dokumenteerimine, selle protsess, kasulikkus ja kulu analüütikute seas. Probleemi lahendamiseks, teeb autor intervjuud analüütikutega ning analüüsis tulemusi kvalitatiivselt. Antud tulemuste põhjal kirjeldab autor üldist dokumenteerimise protsessi, mida järgida ning millest sõltuda kui tekib küsimus dokumendi formaadi ja detailide üle. Lisaks, tuleb välja et peamine põhjus dokumenteerimiseks on arenduse toetamine ning sellele läheb aega kolmandik nädalast. Kirjutatud dokumentatsiooni lugemisele läheb küll väike protsent, kuid töös on kirjeldatud olukorrad, kuidas dokumentatsioon, mis on nii lühike kui võimalik ja detailne kui vajalik, aitab säästa kogu meeskonna aega. Kokkuvõttes, peab iga tiim kokku leppima eelkõige enda seas kui ka kliendiga, kui palju ning mida dokumenteeritakse. Autor toob töö lõpus välja toonud mõned soovitused, mida järgida.

Lõputöö on kirjutatud ameerika-inglise keeles ning sisaldab teksti 50 leheküljel, 7 peatükki, 7 joonist, 4 tabelit.

List of abbreviations and terms

DPI	*Dots per inch*
TUT	Tallinn University of Technology
Stakeholder/client/customer	A person or group of people who are responsible for business requirements and who pay the bills of the software team
An analyst	A person in the team who is responsible for satisfying business requirements by giving input to the developer about the necessary details
MVP	Minimum Viable Product
UML	Unified Modelling Language
XSD	XML Schema Definition
RE	Requirements Engineering
Task	Small (2-3 days) piece of a feature, which provides business value and includes a description for a developer to implement it. Maintained in JIRA [2]
Wiki	A database for collaboratively maintaining documents, notes, requirements, and solution about the system. Maintained in Confluence [2]

Table of contents

List of figures

List of tables

1 Introduction

In the recent decade, the word "Agile" has become popular among teams who continuously deliver new software to different stakeholders. Mainly due to the flexible framework for satisfying business-critical needs by developing software incrementally based on short iterations. In addition, Agile emphasizes the communication inside the team and with the customers in order to avoid exchanging information through long documents and over-complex diagrams [3]. On the other hand, the concept of traditional Requirements Engineering(RE) strongly suggests maintaining knowledge in detailed documents to avoid loss of know-how within the team and between team and stakeholders [4]. This is also supported by several publications about documentation. Quite frequently the Agile doesn't agree with the latest and vice versa.

Therefore, problems such as listed in this paragraph may appear. Firstly, proper documentation and modeling take time and analysts tend to rely on tacit knowledge, whilst the stakeholders are expecting the team to deliver as much business value as fast as possible [5]. Secondly, derived from the first problem, analysts tend to choose quantitative delivery over qualitative, which may cause inappropriate architecture, especially in large and long projects [6]. If requirements and the technical solution is not either documented correctly nor kept up to date, then the risk of losing valuable knowledge in team increases [7]. Whilst the Agile supports using user stories and use cases as documentation, then often the insufficiency of those may become harmful to the team in the long run. Also, some of the post-its used as tasks are not digitalized, which as well contributes to the problem of losing information.

Now, when team members leave or join the team, then without a trace left behind, the new person encounters obstacles in seeking information about working systems [8]. He or she can rely on face-to-face communication with existing team members, but there is no publication that has guaranteed whether this is the most efficient alternative.

In conclusion, documentation in the everyday work of analyst in an Agile team is beneficial, but it always has a cost. Therefore, the theoretical part (Chapter 2-3) of this

paper re-investigates the true purpose of documentation in general and in terms of traditional RE and Agile RE. It is known that traditional RE is more time-consuming and Agile RE can be more efficient in the shorter perspective. Based on existing literature, several attributes to measure cost, benefit, and quality will be explored. The author believes that there exists a happy medium between proper documentation and constant Agile delivery of new features. They can be perfectly combined with each other. One of the main goals of this paper is to find that happy medium.

The empirical part (Chapters 4-6) of the thesis takes a deeper look into the work of analysts (the people who usually write documentation) in one IT enterprise to explore the current real-life situation in different ways of documenting and modeling different artifacts of different IT-systems. The processes, format, level of details and time spent in writing documentation and keeping it up to date are examined in this thesis. This is compared with the time spent in reading the documentation and the coverage of systems with documentation.

As a result, the author finds a solution, which is close to the ideal in terms of effort and impact ratio. The effort is measured with time-consumption and the impact is based on the benefits of the documentation.

To achieve that, an inquiry in the shape of interviews is conducted among the analysts in IT company based on the guidelines of Case Study Research. [9]. The purpose is to get qualitative data from different teams and different projects in both private and public sectors. The answers are being recorded and transcribed into digitalized text, which is then analyzed with RQDA software [10].

Finally, through a discussion in Chapter 5, ideas for analysts to find a balance between investing resources in proper detailed documentation and implementing new features for the system are proposed. The offered solution is expected to be as close to the ideal in terms of effort and impact ratio mentioned in the previous paragraph. This is the gap which has not been solved based on the existing literature and is aimed to be filled in this thesis.

2 Related work

As Klaus Pohl and Chris Rupp write in their book Requirements Engineering Fundamentals that documentation, in general, should follow certain standards such as being agreed upon, unambiguous, necessary, consistent, verifiable, feasible, traceable, complete and understandable [4]. This book also gives a decent overview of the core structure of proper documentation and for which tasks it can be used for. For example, planning, architectural design, implementation, test, change and contract management, system usage and maintenance.

In comparison, in 2001 Agile Manifesto was created [1], which is the basis for a very thorough publication written by Pekka Abrahamsson, Outi Salo, Jussi Ronkainen and Juhani Warsta [3]. One of the central values of Agile is Working software over comprehensive documentation. They elaborate on this by claiming that documentation should be lessened to an appropriate level. One of the purposes here in this thesis is to analyze the "appropriate" level in the context of format and details. In the next chapter, deeper look into different Agile methods and their usage are looked at.

It is common to combine the above-mentioned two methods based on the needs of a specific team and of course the context. When it comes to using bits and pieces from either one of them, challenges may arise. Quite a lot of literature write that describe the latest. It is clearly brought out that requirements documentation is one of the problems of traditional RE due to extreme time-consumption [11]. It can be overcome by using user stories for instance. In addition, the user stories do not tend to change over time. On the other hand, one of the challenges in Agile RE is minimal documentation due to the reason that user story and perhaps backlog are the one and only documents left behind after delivering the new feature. They also find that user stories satisfy system or product features, not non-functional requirements. The high usage of user stories for requirements specification is also supported by 19 out of 24 investigated studies [7].

Similar inconsistency comes out from another study, where the results state that on the one hand, Agile supports a better understanding of requirements due to the immediate access to customer and direct communication [12]. On the contrary, it is found that insufficient format of the user stories makes properties such as consistency and verifiability difficult to validate. Especially in the large and complex software systems

and requirements, code and tests change over time. They offer a solution where after the initial requirements elicitation, final solution must be properly documented using the principles of traditional RE approach. This should mitigate the knowledge loss, but then again it has a cost to the stakeholder and the team.

In last year, another publication is written about Agile quality requirements engineering challenges, where 17 participants from different organizations were interviewed [13]. The results show nine challenges including losing the architectural overview. This means that due to the rapid changes in requirements and minimal documentation which cause isolated knowledge, systems become less understandable and maintainable.

Lastly, when speaking about different challenges - change management plays an important role in terms of documenting essential requirements versus documenting as much as possible. Change tends to be the built-in process of Agile [1], [14]. The precondition for change management is requirements management. Because of that, it is needed to have a link between requirements and the source and even the code. Agile offers Extreme Programming with user stories, but they seem to fail as a baseline for requirements management.

Since this paper focuses on one challenge – documentation in Agile teams – then the next paragraphs are related explicitly to that. In 2014, a very thorough article was published about the cost, benefit, and quality of documentation among software practitioners [15]. Based on existing studies they summarize the attributes for measuring the cost, benefit, and quality of a document. These are used in this paper during the interviews and in the survey. This publication also addresses different formats of the document. A quite interesting finding is that from all (69) of the studies under investigation, only 18% relate to the cost of documentation, whereas 71% connect to quality and 54% to benefit. Therefore, one of the goals here is to measure the cost and analyze whether it's worth it. There are many studies about maintenance and development aid as benefits and completeness and up-to-datedness as qualities [15].

Another type of publication about documentation is based on the survey and interviews conducted within different organizations [8]. They bring out different results based on online-survey and interviews from information seeking and documentation point of view. They also observe some of the respondents after the interviews and illustrate the

differences of reality versus initial opinion. For example, they assess that 32% of the time goes for coding and only 4% on communication. It turns out that the corresponding numbers by observing are 12% and 22%. Based on the same diagram, then only 5% of the time goes for documentation. In addition, a majority of respondents agree that the quantity of documentation in all aspects is either rudimentary or acceptable, not very detailed. When it comes to the level of documentation and satisfaction with search results while seeking information, then in most of the cases people evaluate documentation very important (over 75%), but on the other hand, only around 40% document particular information.

A phrase “limited to 100 words” is in use as a recommendation to describe the product architecture overview and goals for upcoming release [16]. This is against the challenges and recommendations that author was writing about earlier.

In conclusion, there are publications on the topic and some of them agree and some have disagreed with each other. In this paper, the author tries to fill the gap of maintaining knowledge in the software team in a way that it would compliant with Agile principles as much as possible, because of the need to satisfy the client.

3 Background

This existing literature-based section briefly describes the body of knowledge, which is relevant to the approach of the current research. Firstly, the fundamental basics of Agile are described. After that, one sub-part of development is taken into focus – the analysis before development and together with that always comes requirements engineering and documentation. After mapping down the major challenges which come from either traditional or Agile RE, documenting itself is taken apart in terms of benefit, cost, and quality. The frequency, format, and thoroughness of documentation depend on the agility of the team. Therefore, the concept of traditional requirements engineering is introduced and then in comparison, the basics of Agile RE are brought out to the reader.

3.1 The concept of Agile software development

"The "Agile Movement" in software industry saw the light of day with the Agile Software Development Manifesto published by a group of software practitioners and consultants in 2001". Shortly said, the core characteristics of agile methods are simplicity and speed. The development team should concentrate only on delivering functions needed immediately and this must be done fast. The Agile team is constantly collecting feedback from the client and reacting to received information accordingly. They decently answer a question "What makes a development method an agile one?". The method can be considered Agile if it's incremental (small software releases with rapid changes), cooperative (customer and developers or analyst are constantly working together), straightforward (the method itself is easy to learn and modify) and adaptive (able to make last moment changes) [3]. Next, the author writes about some basic and well-known methods of Agile software development.

3.1.1 Methods

This sub-chapter assumes that the reader has some knowledge about the Agile framework. Therefore, the different Agile methods are not elaborated to the details but shortly described with main roles (responsibilities) and practices included within each method.

Extreme programming(XP):

Extreme programming first started in 2001 as a “simple opportunity to get the job done”. It is mostly based on short incremental iterations to deliver a complex system in the long run. XP consists of five phases: Exploration, Planning, Iterations to release, productionizing, Death [3].

The roles are divided between the programmer, customer, tester, tracker, coach, consultant, and manager. Main practices defined in XP are planning game, small/short releases, simple design, testing and refactoring, pair programming, collective ownership, continuous integration, 40-hour week, on-site customer, coding standards and open workspace. The author believes that this XP together with Scrum, which will be described next describe the majority of the fundamental content of Agile [3].

Scrum

The term 'scrum' originally derives from a strategy in the game of rugby where it denotes “getting an out-of-play ball back into the game” with teamwork. Scrum concentrates on how the team should work in order to produce flexible, productive and adaptive systems [3].

Scrum consists of pre-planning phase, where the product backlog list and high-level design of the system are created. It is followed by development phase or phases and is finalized with post-game phase [3].

The roles are Scrum master, product owner, scrum team, customer and management. The main practices are effort estimation, sprint (iteration in XP), sprint planning meeting, sprint backlog, daily scrum and sprint review meeting. Scrum dictates that the length and scope of the sprint cannot be changed after the content has been agreed on [3].

Crystal family

The Crystal Family of methodologies includes a different number of methods and selecting the most suitable one for every project, depending on its size and criticality. There are certain values that are common for each method – incremental development cycles and emphasis on communication and cooperation of people. There exists Crystal

Clear, Crystal Orange, and Crystal Orange Web. They mostly differ by project, development team size, and tools [3].

They all suggest progress tracking by milestones, direct user involvement, automated regression testing, workshops for products and two users viewing per release as policy standards. Crystal Clear also emphasizes using user stories, whereas the Orange requirements documentation. They also say that team itself is responsible for the delivered product [3].

The main roles they mentioned are a sponsor, senior designer, programmer, designer-programmer, UI or database designer, usage expert, technical facilitator, business analyst-designer, architect, mentor, and user. They all can be divided into sub-roles. Besides that, the Clear is for one team and the Orange is for multiple teams [3].

Staging, revision and review, monitoring, parallelism for multiple teams, holistic diversity strategy, methodology-tuning technique, user viewings, reflection workshops are known as the main practices of Crystal family methodologies [3].

Feature Driven Development(FDD)

FDD does not focus on the entire development process, but mostly building and design phase. It is designed to work together with other Agile methods and claimed to be suitable for critical systems [3].

It consists of five phases, where design and building are carried out: developing an overall model, building a features list, planning by feature, designing and building by feature [3].

The roles are divided into key roles, supporting roles and additional roles. Key roles are a project manager, chief architect, development manager, chief programmer, class owner and domain expert. Supporting roles are comprised of release manager, language/lawyer guru, build engineer, toolsmith, and system administrator. Three others that are always needed are testers, deployers, and technical writers. One team member can fulfill the jobs of many roles [3].

The rational unified process(RUP)

RUP is developed to support UML. The core of RUP is defined in a matrix which consists of phases and workflows, where the latest is taking place during phases in parallel. Phases are inception, elaboration, construction, and transition. The workflows are business modeling, requirements, analysis & design, implementation, test, configuration & change management, project management, and environment [3].

RUP defines thirty roles and the casting is rather conventional with the exception of roles defined for business-modeling and environment phase – business-process analyst, business designer, business-model reviewer, course developer and toolsmith [3].

The main practices are to deliver software incrementally, manage requirements, use component-based architectures, visually model software, verify software quality and control changes to software [3].

Dynamic Systems Development Method(DSDM)

The fundamental idea behind DSDM is that instead of fixing the amount of functionality in a product, and then adjusting time and resources to reach that functionality, it is preferred to fix the latest, and then adjust the amount of functionality accordingly [3].

It consists of five phases: feasibility study, business study, functional model iteration, design and build iteration, and implementation. First two are done once, last three are done incrementally [3].

Main roles are (senior) developers, technical coordinator, ambassador and/or adviser user, a visionary, and an executive sponsor. It includes active user involvement, empowering teams to make decisions, focusing on frequent delivery of products, designing to fit for business purpose, making changes reversible, baselining requirements at a high level, testing throughout the lifecycle, collaborative approach shared by stakeholders [3].

Adaptive software development(ASD)

Fundamentally, ASD is about “balancing on the edge of chaos” – its aim is to provide a framework with enough guidance to prevent projects from falling into chaos. It is carried out in three phases: speculating, collaborating, and learning. The names should emphasize the change in every phase. ASD is more component-oriented than task-oriented, which in practice means focusing on the quality of the tasks rather than the tasks itself. ASD is also

known to be mission-driven, time-boxed, change-tolerant, risk-driven and iterative as other Agile methods [3].

When it comes to the roles, then ASD originates from organizational and management culture, especially from collaborating teams. However, this does not describe team structures in detail. An execute sponsor is responsible for overall project and some participants in application development session are only listed [3].

It does not offer many day-to-day practices. The main problem with ASD is that most practices are described as what should be done, not could be done [3].

There is also an Open Source Software Development, which is not relevant to this study since it mostly focuses on the work of volunteers. In addition, *Kanban method* is being described [17].

The goal of the Kanban method is to maximize the workflow and shorten the average time to complete one item by limiting the amount in progress. Kanban board is widely used to visualize the process by using several columns, while each of them representing a stage in the software development process. The workflow will progress, because psychologically a person does not want to keep the one column full of items, but to move them to next column [17].

Kanban is relatively new to software engineering but it has existed in manufacturing for two decades. Kanban is similar to Scrum but does not define certain roles, the length of the sprint is not fixed, and changes are allowed to be made during the process.

As can be seen, all methodologies have some core similarities and at the same time several differences. In the end, it all depends on the team, the project, the stakeholder and the cultural background which method is being used. All of them agree on 4 basic Agile principles [3]:

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

Now, the content dives deep into one sub-part of Agile development – the analysis, which is connected with the documentation and requirements engineering. It does not matter if it will take hours, days, weeks or months, a bit of analysis in some format needs to be done before a programmer can literally start writing code. In Agile methods, it is done continuously in short (2-4 weeks) cycles, whilst the traditional Waterfall or V-Model aim to do it for the whole system at once [4].

3.2 The basics of traditional Requirements Engineering

This section is based on the book "Requirements Engineering fundamentals, 2nd Edition" written by Klaus Pohl and Chris Rupp [4]. They write a very solid concept of RE and some parts of it are relevant to this paper.

According to past studies, 60% of all errors in development projects come from the phase of requirements engineering and they are discovered once the project has been deployed to a production environment. Developers tend to implement what they believe the incomplete requirements claim to be saying. The cost of fixing the errors in final step is 20 times higher than in the phase of requirements engineering. In order to discover faults and gaps later in systems, the concept of requirements engineering will be introduced [4].

In order continue the word "requirement" is being defined. It can be either

- A condition or capability needed by a user to solve a problem or achieve an objective.
- A condition or capability that must be met or processed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.
- A documented representation of a condition or capability as in (1) or (2).

The phrase "Requirements engineering" is defined as follows:

- It is a systematic and disciplined approach to the specification and management of requirements with the following goals:
    - Knowing the relevant requirements, achieving a consensus among the stakeholders about these requirements, documenting them according to given standards, and managing them systematically.

- Understanding and documenting the stakeholders' desires and needs, they specifying and managing requirements to minimize the risk of delivering a system.

The traditional RE is defined with the following four core activities:

- Elicitation, where with different techniques the requirements are obtained from the stakeholder and other sources.
- Documentation, where the elicited requirements are described adequately.
- Validation and negotiation, where it is being guaranteed that predefined quality criteria are met and documented requirements are validated and negotiated early on.
- Management, where it is made sure that any measures that are necessary to structure requirements and prepare them so that they can be used, maintained and implemented are taken.

The person or shall it be said requirements engineer must be capable of analytic thinking, be emphatical, have good communication conflict resolution, and moderation skills, be persuasive and self-confident.

3.3 Challenges of Agile RE and traditional RE

There are many challenges which the teams encounter on the daily basis when trying to follow either Agile or traditional RE principles. For instance, customer availability, budget and schedule estimation, inappropriate architecture, neglecting non-functional requirements contractual limitations and requirements, requirements change and change evaluation, minimal documentation and reliance on tacit knowledge, insufficiency of the user story format, difficulties in the prioritization of requirements, imprecise effort estimates, team maturity, and coordination between teams [7], [11]- [14], [18].

In this paper, one challenge and its consequences are examined – minimal documentation and relying on tacit knowledge. Yes, it is up to the team itself how often and how much and what to document, but since the golden rule does not exist based on the investigated body of knowledge, this paper tries to figure that out. Whilst traditional approaches try to produce enough documentation to answer all the questions that may arise in the future, then the writer or analyst needs to anticipate those questions in concise and

understandable manner. This may lead to writing too much documentation. On the other hand, the analyst should not spend much time on producing documentation in order not to waste valuable time. [18]

There are consequences to the minimal documentation such as miscommunication between a software delivery team and the customer, especially in large projects and different geographical locations [11]. A similar problem may appear if teams are big enough to not work in the same room nor building. Then again, the problem of finding enough time for that still exists.

Figure 1 below presents one option to view usefulness and amount of documentation. [19]

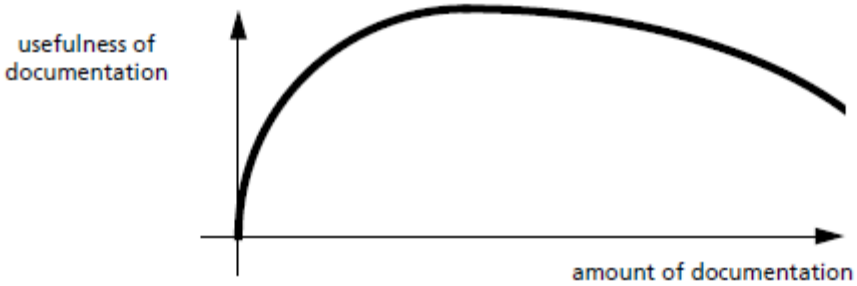

Figure 1. Amount and usefulness of documentation.

In order to evaluate the validity of this graph and give a solution to the challenge stated above, the next chapter describes documentation more closely.

3.4 Documentation

Use of information or also known as documentation comes from seeking of the information, which is derived from the need for information. Those are based on individual's internal cognitive structures, emotional dispositions, and affective reactions, which are dictated by their current environment. Documentation is just one form of information use that is under focus in this paper [8].

The documentation itself is a wide notion, especially in software development. Therefore, several definitions are brought out to understand the concept [15]:

- Documentation is a written description of software systems.
- Documentation is expected to provide precise information about the systems.

- Documentation can refer to the product manual that developers created for non-developer users.
- Documentation is created for communication among software engineers.
- Documentation can refer to different artifacts, including requirements, design, code, code comment, test cases etc.
- Documentation can be presented in different formats, such as traditional text, graphical models (for example UML) or dynamic hypertext systems.

In a decent article written in 2014 by several authors, the quality, benefit, and cost attributes of a document are concluded and listed in Table 2. Those attributes do not apply only to documentation in the context of requirements engineering, but software engineering in general. This chapter is mostly based on this study and the attributes of documentation cost, benefit and quality are brought out. [15]

Table 1 below sums up the pre-mentioned attributes of a documentation.

Table 1. The benefit, quality and cost attributes of the documents.

Benefit attributes	**Quality attributes**	**Cost attributes**
Development aid	Accessibility	Development
Management decision aid	Accuracy	Maintenance
Maintenance aid	Author-related	Usage
Architecture comprehension	Completeness	Document size(length)
Code comprehension	Correctness	
Perceived importance	Information organization	
Reduction in effort	Format	
Actual usage	Readability	
	Similarity	
	Spelling and Grammar	
	Traceability	
	Trustworthiness	
	Up-to-date-ness	

Under Related work (Chapter 2) it is stated that over 75% of organizations under the study value the documentation as important, while under 40% of companies document particular information [8]. The gap between them can be connected to the following artifacts: documenting solutions and instructions, knowledge about application domain,

decisions concerning alternatives in implementation, specifics/lessons learned, general view as architecture/design model and concrete implemented solution details (code). The popularity of the usage depends on the organization.

3.4.1 Documenting based on traditional RE

If chapter above summarizes documenting in general, then when going deep into requirements engineering in a traditional way, then even more details can be documented.

For example, the introduction of the document should consist of the purpose of the document, system coverage, stakeholders related to the document, definitions, acronyms, abbreviations, references and a short overview [4].

Then there's General Overview section, which should contain system environment, architecture description, system functionality, user and target audience, constraints, and assumptions.

In order to stay in scope for this paper, no more details are described. What is important, is the level of details. If a person in the Agile project has enough resources to document the requirements or solution, then he has a proper model to do so - following the quality attributes and concept of requirements engineering.

3.4.2 Documenting based on Agile principles

The summary of the previous chapter does not explicitly agree with one of the four core values of Agile Manifesto [1]: Working software over comprehensive documentation. It is seen as infeasible or at least, as not cost effective. The scope of the documentation is quite often limited and focuses only on the core aspects of the system [18].

Interestingly, Extreme Programming, Scrum, Rational Unified Process and Open Source Development state that documentation should be done in the very last phase of the development, where the whole system has been deployed into production and no more new features will be delivered for the customer. None of them specify, what if the MVP of the project has been delivered, but the client requires more features. The question, who updates the documents and what should be modeled or documented throughout the project is not explicitly defined.

Crystal family and its methodologies say that progress should be tracked by milestones, not with documentation. There's also a difference whether they talk about Crystal Orange or Crystal Clear method. The first demands UI design documents and inter-team specification, whilst the Clear only screen drafts and design sketch [3].

None of the methods specify, which role should be responsible for documentation, except Feature Driven development and Crystal family. In FDD Build engineer or Technical Writer are taking care of it. The first manages the publishing of documentation, the last one prepares user documentation. In Crystal family, a role of the writer is being mentioned to be responsible for documents. For the sake of simplicity, the further content will use the word "analyst".

It is needed to understand that the best documentation should not be an excuse if the project is supposed to deliver software, but fails to do so. However, this does not mean that it is generally unimportant. As a balanced solution, the documentation needs to be light-weight enough that it is easy to read and write but thorough enough to not give false information to the reader [19]. The final decision what to document in Agile teams comes down to the team and project. If the customer and the team have decided what is needed to document in the project, then this decision should be followed and again follow-upped later.

4 Case Study design

This chapter explains how the case study is designed in order to solve the problems stated in previous chapters. The study itself is done to fill the gap described in Chapter 2 and Chapter 3. The identified gap is taken under investigation for both academic and industrial purposes. The author believes that the results of this case study can be a great contribution to the software industry.

The objective and a purpose of the study is to investigate if and how the Agile methods can coexist together with principles of proper documentation point of view. The second purpose of this study is to formulate a conclusion how it can be done with the lowest cost for the client and the software team in the long term.

Due to the limited length and scope, this study is classified as single-case and holistic, because most of the software teams are using bits and pieces from Agile and traditional software development principles from the standpoint of documentation.

4.1 Research questions, propositions, and hypotheses

Based on everything that is written in previous chapters, the following is known

- Agile principles do not emphasize writing long documents.
- Agile principles emphasize continuous delivery through incremental development cycles.
- Proper and detailed documentation tend to be rather long than short when it comes to the number of pages.
- Those documents are needed to be structured in order to maintain knowledge in the team in an understandable way.
- The balance in finding time for second and fourth statement is not known.

Therefore, the following research question and sub-questions are solved with this study:

- How to write proper and detailed documentation in software development team, whose work is based on Agile principles?
    - What format, processes and how deep details are used in order to write documentation in Agile projects?

- What is the most efficient balance between writing and keeping documentation up to date versus continuous delivery of new features for the stakeholders in terms of time-investment?
- When is the best time to write documentation for the software project?

Affiliated from research questions the following propositions and for some hypotheses are stated:

Table 2. Propositions and hypotheses

Number	**Proposition**	**Hypotheses**
1	If the concept of Agile or any other model would offer the perfect solution, then all the analysts would be using it	
2	If writing proper documentation would not be time-consuming then everybody would do it	Writing proper documentation is considered mostly as time-consuming activity
3	If one model or method of software development would lack something then, it would be replaced by a segment of an another model	The majority of respondents use a mix of different models
4	If there would be always certain format, structure or method of documentation defined for every project in any size, then the corresponding person would use this	The format, methods, and structure of documentation depends on the time-management skill of the analyst
5	If the client or customer requires another feature to be built into the system, then in most cases according to the previously defined contract, a team would do so	
6	If there is a client who pays for the product to be delivered, then analyst would	Analysts tend to choose satisfying business needs

	fulfill the requirement as fast and with high quality as possible	over writing proper documentation
7	If an analyst would have to write proper documentation, then he would do it within work time	None of the analysts work overtime to write documentation
8	If an analyst would have certain time span weekly or monthly to write proper documentation then it would help to maintain knowledge in the long run	

4.2 Case and data selection

The purpose of the embedded case under investigation is to get knowledge about the documentation processes, then the format about documentation and lastly the time-investment in it. In order to gather proper data, a survey with open questions is created. The supported questions can be found in Appendix 1. Based on the questions, interviews are conducted among the analysts in a software company (Helmes AS) which is today offering consulting and software development service for business-critical problems. The work of this company is based on 34 self-organizing teams (altogether 252 people), where most of them are currently profitable. The non-profitable teams are considered as just started. The company was founded in 1998 and the quarterly turnover is a little bit over 7 000 000€. [20].

Becoming more detailed, the respondents in the survey and interviews are not all employees, but the analysts or consultants for product owners since the company mostly follows Scrum (Section 3.1.1). The analyst is a person between the client and the developers. On the one hand the analyst must perform business analysis, but on the other hand, he or she must also have a fairly decent understanding of the technical details of software development. The following are the expected tasks for the analyst in everyday work:

1. Eliciting requirements according to the client's need. Also, the analyst should be ready to offer alternative solutions based on the technical complexity and business case priority.
2. Analyzing those requirements to the level where they can be estimated and developed.
3. Making sure that the solution is stored in documents, diagrams or different models.
4. Understanding the business in order to guide the programmer during development.
5. In some cases, testing the solution before deployment.

Every team in Helmes should have 1-2 analysts so the expected pool of participants in the interviews is 6-8 people. There is not a study to be found, which can relate to this topic in a way that it can be replicated 100%.

4.3 Data collection procedures

As mentioned the data is conducted through interviews. Interviews are supported by the questions derived from the hypotheses in Section 4.1. The author sent a call to all the analysts in Helmes so that volunteers who liked to contribute to solving the research questions could do that. Face-to-face interviews lasted for 30-45 minutes depending on the interviewee. Interviews are recorded and then transcribed into digital text, which is qualitatively analyzed with RQDA software. The purpose of the interview is to get data about different ways of documenting in software development teams, whereas the work of the team mostly depends on the stakeholders and their requirements. The link to the audio and transcribed text files is in Appendix 2.

4.4 Analysis procedures

Based on the answers from the interviews and the help of RQDA the author performs qualitative thematic analysis on the results [10]. The goal of the analysis is to find information about the following:

- What Agile methods were used in the teams
- Processes and methods used for documentation and keeping it up to date
- Different formats of documentation
- Time spent on reading and writing the documentation
- Time spent on delivering new solutions to the client

- The balance and correlation between cost and benefit of documentation

The results are summarized, structured and compared with the theoretical framework in Chapters 2-3. Based on that, it is evaluated whether the teams are compliant with agile methods or not, what documentation principles are followed and what are the correlations between those. This is done through pattern matching and explanation building [9].

Finally, a proper summary through discussion is offered to the reader of this paper with answers to the research questions and (dis)approval of the hypotheses.

4.5 Validity procedures

In order to guarantee the credibility, transferability, dependability, and confirmability the following measurements were used [21]:

1. The researcher has been studying Business Information technology for 3 years and is working in the software development company.
2. The respondents were expected to participate on their own initiative to make a contribution to the corresponding field. Therefore, the honesty is guaranteed.
3. Before the implementation of the designed case study, debriefing with the author's supervisor and mentor at work are done.
4. Creating a pilot interview among 1-2 analysts to get feedback about the formulation of the survey in order to improve it for other participants.

5 Results

The results are analyzed qualitatively by first becoming familiar with the data, then generating initial codes, which are the basis for initial themes [9]. Next, the themes are reviewed and finalized. The themes and frequency of theme-related code occurrence are available in Table 3. Each of the themes is described in more detail in the sub-parts of this Chapter 5.

Table 3. The themes with occurrences of the analysis.

Final Theme	**Frequency of occurrence**
Being Agile with different methods	74
Role of an analyst	16
Documentation cost and benefit	36
Format and details of documentation	27
Process of documentation	29
Updating documentation	10
Usage of tasks and wiki	53

Most of the themes are in correlation with the questions asked in the interviews (Appendix 1). “Being Agile with different methods” have the most occurrences because every used Agile method is used as separate code, while the Updating documentation is related two only one code. When the author uses terms “task” or “wiki”, then based on the interviews, JIRA and Confluence made by Atlassian are the tools used in every team [2]. JIRA and Confluence are also meant in the next parts of the thesis, if the researcher says “task” or “wiki”. In the next sub-chapters the summary and short discussion of the themes are written.

5.1 Respondents being Agile

Here, it is shortly described the respondents and the way of how they consider themselves working in Agile teams. Eventually, the author managed to conduct interviews with 7 analysts and 1 team lead from 8 different teams. The work experience of the respondents is from half a year to 5 years and they all say that in general, their job is to be a "link" or a "middle-man" between the customer and the development team. Mainly, the detailed role of an analyst is the same as written in Chapter 4.2. Most of the teams have one analyst, who is responsible for both business and system analysis. One team has system analyst separately, who is also a former lead developer. Also, a couple of teams have a separate analyst from client side with who the analysis of software systems is done in cooperation. The majority of documentation is done by analysts.

All of them confidently acknowledge that they are working in Agile teams, except one interviewee, who has doubts "We have had problems with increased scope, which is not very Agile. We did not deploy something small with value into production".

When it comes to the usage of different Agile methodologies consciously, then none of the teams have taken one certain concept and follow it. It turns out that the teams are deliberately working in 1-2 weeks flexible sprints while engaging the customer and managing the process on either Scrum or Kanban board, which are described in section 3.1.1. Some of them mention different practices such as code review, sprint planning, retrospective, user stories, stand-ups, continuous deployment, planning poker, pair-programming, time-boxing, analysis workshop prioritization with the client, but that is all. The list seems fairly short though compared to the methods summarized in section 3.1. The majority takes the combination of methods into use on the go based on the current need and experience. As only one example, one analyst states: "During the time the methods have changed a bit. We did not have a continuous deployment at first. Everything is constantly in the change and we adapt depending on the situation". Therefore, hypothesis number for proposition number 3 is proved and assumption for the main research question exists. Figure 2 illustrates the Agile methods and frequency of occurrence. It has to be said, that actually the teams are using more methods, but they just could not think of all of them during the interview, because the methods are considered too "normal" part of work.

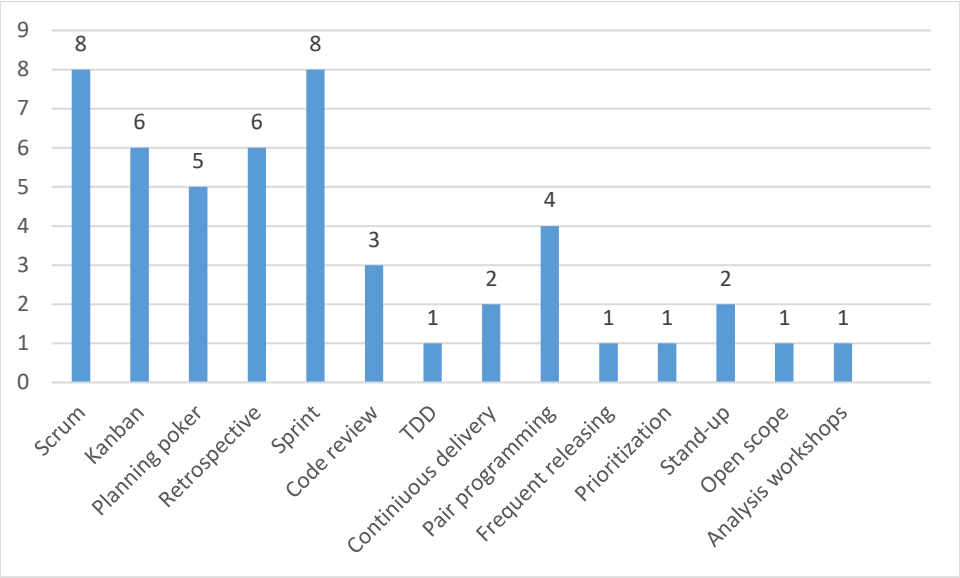

Figure 2. The frequency of mentioned Agile methods.

In addition, the product backlog for the teams, of course, varies through time, but the average of estimated hours ready for development is approximately one month and one week. The aimed average size of one task or story is 2-3 days.

5.2 Process of documentation

Firstly, the process of documentation depends on many things such as the scope of the requirement, team size, agreement with the customer, the customer being in public or private sector, habits and innovation in the team. For example, one analyst says: “Documentation is not extra activity, it’s part of the work and it is done during analysis. I always do it on the go”. Then, there were others who say the opposite. Based on all the answers and acknowledging that the details differ, the results for generic documentation process is presented as business process diagram in Figure 7 which is in Appendix 3 due to the size. The main difference between teams is the amount of repeating the activities and time spent on one. For modeling author uses Academic Signavio [22].

As can be seen, the flow cannot begin if a developer does not have the task assigned by the analyst. This is the most important precondition with what all the analysts agree with. Furthermore, the first parallel gateway describes the way of eliciting requirements from the customer. It can be only one or all of them. After initial analysis of the scope and size of the task, analyst either starts creating the first task in JIRA or wiki page in Confluence. Then, after deeper analysis, the analyst specifies the task and/or wiki in parallel with

modeling diagrams or creating prototypes and mock-ups about the solution. Most of the analysts do it together with analysis. After validating the summary with the customer, about half of the analyst structure, correct the documentation with creating links between corresponding tasks and wiki. Then, the development and testing process begins and it only leads to the fulfillment of requirements if something unexpected does not appear during the process.

The green tasks are considered as activities contributing to documentation. The one and only red activity on the diagram is labeled as the essential part of Agile development, which actually creates the final documentation in order to guarantee proper knowledge maintenance in the team. Without it, the information about the requirements and the solution tends to be spread around in many places. Unfortunately, the latest is not done by all respondents, because they claim it is not worth the time investment. Especially, when they cannot get past from the first XOR gateway.

5.3 Format and details of documentation

By saying “format and details”, the researcher means the alternative ways of documentation (pure text, models, diagrams, tables, meeting memo) and how deep technical details are used in writing the description for tasks or overall solution in the wiki.

Firstly, only one of the respondents deliberately has read and practiced the concept of requirements engineering and she is quite satisfied with the results. But none of the others, have heard about it. Interesting is that during interviews it came out that most of them have actually used many practices of RE. Therefore, it can be stated that most of the work is done based on experience and skill to adapt to different situations.

When it comes to the format, then it turns out that all of them use pure text, tables, activity diagrams, sequence diagrams, tables, prototypes, mock-ups altogether. In some cases, one supports another and vice versa. One respondent says that they had even used XSD. It is also discussed that the format depends on the reader. Complex integrations tend to be created in sequence diagrams for developers and business process or activity diagrams for the customers. Though, it is not within the scope of this thesis to find the correlation between the popularity of format of the documentation and the business requirement.

The conversation about the details varies based on their habits and shaped work processes. Details can reflect only business requirements about the system or on the other hand the length and data type of the field on the certain pop-up window. One analyst tells: "Documentation should be done as less as possible and as much as needed". Whilst the other explains: Since it is public sector, then the documentation must be very detailed." Teams working for a private sector tend to stick with fewer details though, except for the case where the documentation is the one and only tool of communication due to the reason that client's development team is located in another country. Table 4 below concludes the cause-effect relationships that came out during the interviews.

Table 4. The cause-effects of details of the documentation.

Cause	**Effect**
The experience of developer differed in the team	The details of documentation tended to depend on the developer for who the task was assigned to
The client was from either public or private sector	The details strongly depended on the sector where the client was from.
The target group for reading the documentation differed	The details depended on whether the reader was a business stakeholder or a developer. In the first case, the analysts tended to stick with activity diagrams. The second case includes complex sequence diagrams. Especially in the context of integrating systems
Systems had changed and documentation may have gotten outdated	The details were written on a high-level, hence the systems changed, the documentation had not needed an update.
Theories about proper documentation were long and time-consuming	The level of details was only based on the experience
Creating a thorough class and object diagrams were only done at the beginning of the project	They tended to stay outdated due to huge time-consumption
The analyst did not have time to write the minor details about every solution	The developer adds them to the task comments during the development

5.4 Keeping documentation up to date

Figure 3 below illustrates the percentage of covered and updated documentation. One respondent does not have enough information about it and another one is working with

such new project that the documentation has not needed an update. Another analyst said that the 100% of the system is covered with documentation, but with JIRA tasks only, which means the essential (colored red) part in the process diagram is not accomplished. So, if a new person should join the team then he would get updated information about the system, but as the analyst also agrees – it is not very convenient.

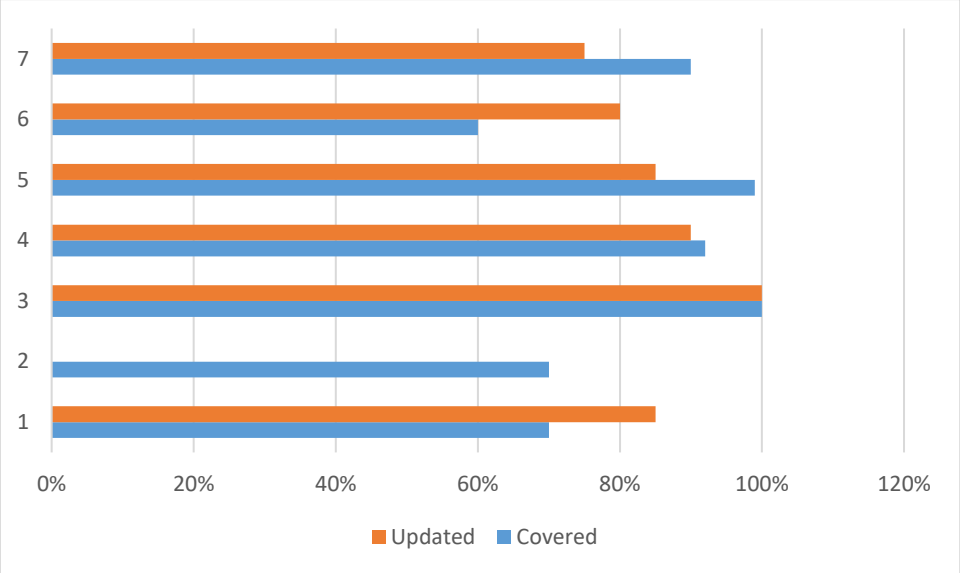

Figure 3. The percentage of covered and updated documentation.

Those are the facts about the current situation. Another thing is the process of updating it. In other words, when and how often? The work process is very individual but based on the data, there are two high-level options for documenting and updating

1. Everything is done on the go together with analysis
2. Separate time is booked on the calendar during the week to follow-up the action points afterward.

Some say that option 1 is less time consuming and they have not used option two because there are many other things going on and booking separate time is impossible. Two analysts even confess that during the day the work is very intensive, so they do it after work time when everybody has gone home. Previously was stated that priority number 1 is to have new work for the developer and then after that, the analyst can focus on documentation. When it comes to updating the documentation, then the analysts would definitely choose writing about new features than updating the old ones.

The environment for updating the documents also varies. For some teams, old solutions are stored in Microsoft Word documents while the others use shared Confluence and then there are teams who only use JIRA task descriptions. In conclusion, there is no right or wrong answer – what is known for sure is that a time for that needs to be found and it depends on the time-management skill and experience of the analyst. Therefore, based on sections 5.2-5.4 hypothesis for propositions 2, 4 and 6 are proved and 7 is not proved.

5.5 Benefit and cost of documentation

Now the "why" question – why analysts write documentation and what is beneficial about it? During the interviews, they were given a list of benefit attributes based on "Cost, quality, and benefits of software development documentation: a systematic mapping". [15]. From that list, they chose 2 most important ones. The results are in Figure 4.

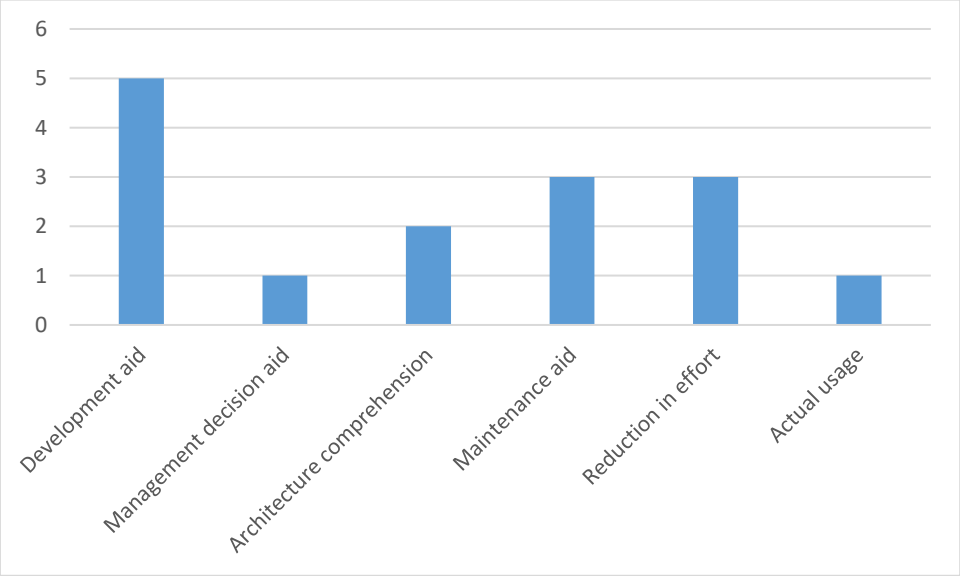

Figure 4. The most important benefit attributes.

Definitely, the results cannot be the only basis to rely on, because the pool of respondents is not vast, but the overall idea that reflected from the answers was that documentation is meant for:

1. Developers, because an analyst does not always have time to answer questions about specifications face-to-face.
2. Customer's representative, a new team member, a new potential customer or 3rd party partner to get a quick overview of the solution. Sending a link to the wiki is more

effective in flexible and intense Agile world than describing the solution many times to different people.

In most of the teams the client expects and assumes proper documentation, but often they do not read it at all. For example, one of the answers is “They did not have any expectations, but maybe in the contract, it was promised to do it. Most probably they expect but they do not check. They sometimes wanted the drawings and diagrams, which they used for understanding the system by themselves.” About the benefit for the developer, it is said: “Ideally developer uses the documentation only, but sometimes the face to face communication is needed. Very rarely developer talked directly with the client and the analyst documented afterward.” To sum up, all the respondents agree that documentation is important to aid corresponding parties in Agile teams. Theoretically, everything should be documented 100%, but due to other priorities, it is almost never like that. One reason for this is the cost of it.

The arithmetical average of the results about the ratio between time-investment into writing and reading documentation by others during one work-week is reflected in Figure 5. When it is said “writing” documentation then it is meant everything colored red in Figure 7**Error! Reference source not found.**. Reading, on the other hand, is more d ifficult to evaluate. During the interviews, it was tried to figure that out by taking into consideration the questions from developers or client and developed features, which differ from the originally written specification. In some cases, it is impossible to evaluate it due to the many of stakeholders.

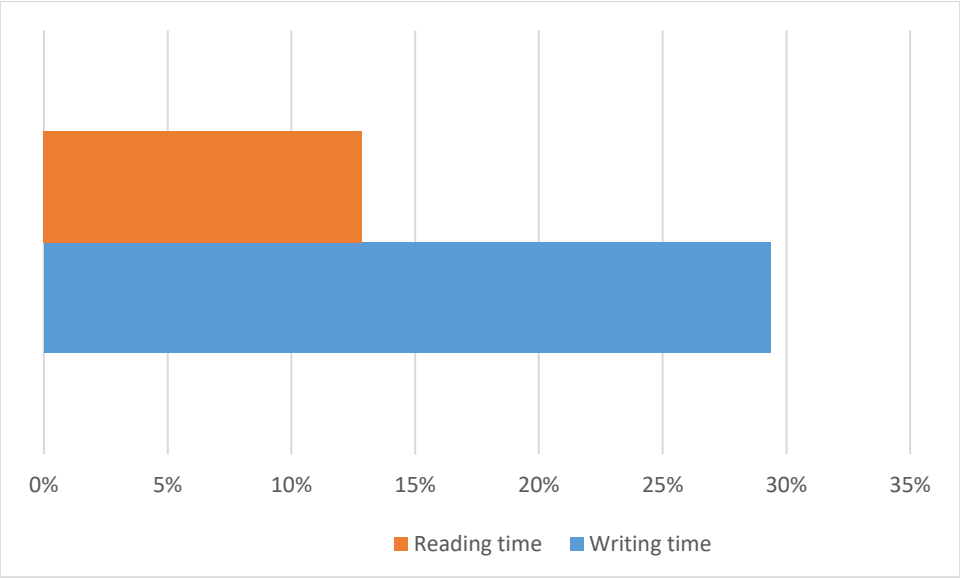

Figure 5. The time investment of documentation during the work week.

The smallest number of hours investing time into documentation us 2 hours and largest 32 hours. Figure 6 shows the relation between time investment, the percentage of documentation coverage, level of details and whether the analysts use option 1 or 2, which are mentioned at the end of section 5.4.

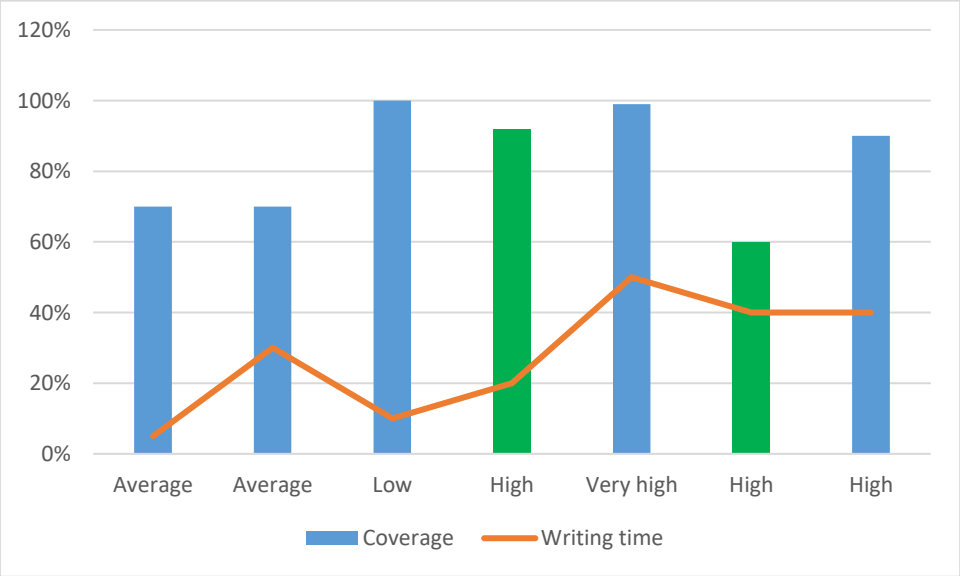

Figure 6. The relation between writing time, coverage, details, and timing.

Green columns use option 1 of the timing of documentation. Level of details is low when analyst only sticks with JIRA descriptions. It is average when he writes documentation with diagrams to the wiki and links JIRA tasks with it. The level is considered high if the content of documentation is very detailed, the majority of the solutions are up to date and

covered with both description and modeled diagrams. One obvious connection between writing time and level of details is that the less time analyst invests in documentation the less detailed it is. Since the percentage of coverage strongly depends on the length and size of the project, then the only relation here is that the coverage does not depend only on writing time. One can invest 10% of the time in a week in the documentation but gets system 100% covered, whilst the other analyst can invest 40% and get system 60% covered, but with more details. The decision, which is more important lies on the shoulders of the working habits of the team and the requirements from the client. According to the one respondent: Also, the developer updates the JIRA with comments when small details are figured out.”. This is definitely a decent manner that contributes to the more detailed documentation with low time consumption.

6 Discussion and future work

In this chapter, the answers to research questions are given together with ideas for the future works for other researchers.

What format, processes and how deep details are used in order to write documentation in Agile projects?

When it is said "format", the author means whether the documentation is written in pure text, using models and diagrams, prototypes or combination of them. "Details" mean if documentation covers purely business requirements or also explanation with data types and key-value pairs about how to fulfill the requirements. It turns out it depends on the audience and the complexity of the task. Since the teams work adaptively in the Agile environment, then the analyst must have enough experience to choose what format to use. One should not forget making the ideas and solution understandable as concretely and clearly as possible.

The core process of documentation is presented in Figure 7. Understandably the exact sequence and repentance of the activities depend on the team and project, but generally, this would be summarized process to follow in the Agile environment. One option for related future articles is to explore the processes and format of documentation in either huge corporations or small start-ups. Process offered here can be categorized as the happy medium between those two.

What is the most efficient balance between writing and keeping documentation up to date versus continuous delivery of new features for the stakeholders in terms of time-investment?

The most important thing is to continue analyzing work to the point where the developer has enough work to do, This rule should be never overridden because satisfying stakeholder's requirements is priority number one among all respondents. If the schedule is not that intense, then wise analyst should use it for documentation. Firstly, it must be guaranteed that the core framework, requirements and transactions are documented in either wiki or task descriptions. It can be also combined with the job of analysis. Instead of just thinking about the solution, it can be either written down or modeled. As can be seen in Figure 6, then it is very difficult to say what is a golden rule in finding the balance.

Rather the team should internally and externally agree on the documentation habit and follow it, whilst taking into account that over-documenting is time-consuming, especially if the percentage of readers is quite low. On the other hand, the knowledge should stay in the team and relying on tacit memories can become time-consuming on its own. An idea for future work can be to observe the analysts in weekly basis and measure the exact time they spend on documenting, analyzing, testing and communication.

When is the best time to write documentation for the software project?

As section 5.4 describes, there are two types of people – those who write documentation with very small increments together with analysis and those who reserve a certain time span at the end of work-day to follow-up the week for example. It is not said that it is only black and white. Rather, there are people who tilted more to one side than another. Overall tendency is that documentation should be written if other responsibilities are handled. The author proposes to practice both methods for future case studies.

How to write proper and detailed documentation in software development team, whose work is based on Agile principles?

The author proposes summarized practices based on the whole paper which can be followed to answer the main research questions:

1. At first, making sure that developer has work to do
2. Documenting the core of the system with low-level details
3. Knowing what the customer expects from the bought service
4. Knowing the time-management skill and the discipline, whether to document on the go or booking certain time during one period
5. Linking wiki, tasks and meeting notes together for the overview
6. Spending 1-2 hours per week on organizing and updating documentation
7. Asking the developer to supplement tasks by leaving comments after decisions about small details
8. Adding “Documentation” column to the Kanban or Scrum board
9. Not getting stuck in old habits and spending some time in exploring new ones

7 Summary

It is known that traditional waterfall method is mostly replaced by agile methodologies in software engineering. Agile Manifesto emphasizes communication, flexibility, and adaptivity, which helps to improve the work processes of software development teams. At the same time, while Agile presents minimalistic documentation, there is a solid body of knowledge about requirements engineering and documentation in general, which claims the opposite of Agile. Combining those two rises many challenges, which are shortly described in section 3.3 and supported by many journal publications. Therefore, the aim of this thesis *Process, benefits, and cost of documentation among analysts in* A*gile software development teams* was to gather data from analysts about documentation in Agile software teams and propose ideas how to improve the described situation.

Based on the gap in Chapters 2-3, the research questions, propositions, and hypotheses are stated, which are a fundamental basis for the case study. To get data for the case study, 8 interviews were conducted with 7 analysts and 1 team lead in Estonian IT company Helmes AS. After recording, the content is transcribed and then qualitatively and thematically analyzed. The questions that were discussed during the interviews are in Appendix 1.

The results are presented in Chapter 5. It is found that analysts in Agile teams tend to work based on the tacit experience and use a mix of different methodologies. For example, they all state that they work in Agile teams, but none of them used only one certain methodology. They tend to use a mix of them together with steps of requirements engineering, but not deliberately. The focus rather goes to delivering new business value, not consciously practicing different techniques, which are described in Chapter 3.1.1.

Even if projects, teams, and customer differ in every self-organizing team, the generic flow for documentation tends to be mutual. It is presented in Figure 7. It is extremely important that the documentation process never starts if a programmer does not have enough work to do. If this condition is filled, then the details, format, benefit, and cost of documentation can be talked about. As with Agile methodologies, the format and level of details depend also on the client's expectation, the size of the task and habits of the analyst to make the thoughts understandable as clearly and fast as possible. One can use a combination of text and diagrams, whilst the other only uses task descriptions to store the

knowledge. It is considered essential to at least a trace about the solution in some kind of form for others.

By arithmetic average, the analysts spend approximately 30% of their work time in documentation and keeping it up to date, which means they see a benefit in it. By documentation, it is meant structuring, correcting, and linking task descriptions, meeting notes and wiki. The majority think that the most useful benefit is development aid. Many analysts tend to do it together with a process of analysis, while the others like to reserve a certain time span at the end of the workday and do everything at once in one week range. It depends on the self-discipline and personal habits.

To conclude everything, then the process of documentation in Agile teams must be agreed upon internally and also with the customer in order to find a solid balance. An analyst can start with documenting high-level description and if he is aware of his schedule and assumptions from developer and client, then this is the fundamental basis to adapt to. If analyst spends 1-2 hours per week to digitalize and update the knowledge, then this is considered as “good enough” for new team members. In some cases, when the integrations or tasks are complex, the analyst must be ready to spend more time on it and the efficiency comes from doing it together with analysis. Commenting tasks and linking them with wiki has a very small cost, but it turns out that a huge impact.

References

[1] J. H. Martin Fowler, "The Agile Manifesto," p. 7, 2001.

[2] Atlassian Corporation, "Software collaboration and development tools," Atlassian Corportaion, 2002. [Online]. Available: https://www.atlassian.com/company.

[3] O. S. J. R. a. J. W. Pekka Abrahamsson, "Agile Software Development Methods:," p. 112, 2002.

[4] C. R. Klaus Pohl, Requirements Engineering Fundamentals, 2015.

[5] Inderscience Enterprises Ltd. , "Documentation strategies on agile software," p. 15, 2012.

[6] B. R. L. C. R. Baskerville, "Agile requirements engineering practices and challenges: an empirical study," p. 32, 2010.

[7] C. S. A. M. L. d. V. Juliana Medeiros, "Requirements engineering in agile projects: A systematic mapping based in evidences of industry," p. 15, 2015.

[8] J. v. G. J. M. D. W. Stefan Voigt, "A Study of Documentation in Agile Software Projects," p. 6, 2016.

[9] M. H. A. R. B. R.¨.¨ PER RUNESON, CASE STUDY RESEARCH IN SOFTWARE ENGINEERING. Guidelines and Examples, John Wiley & Sons, Inc.,, 2012.

[10] "RQDA," [Online]. Available: http://rqda.r-forge.r-project.org/. [Accessed May 2018].

[11] S. i. s. S. M. M. D. S. S. Irum Inayat, "A systematic literature review on agile requirements engineering," *Elsevier*, p. 15, 2014.

[12] C. L. D. D. M. P. Ville T. Heikkila, "A Mapping Study on Requirements Engineering in Agile Software Development," p. 9, 2014.

[13] M. D. Wasim Alsaqaf, "Agile Quality Requirements Engineering Challenges: First Results from a Case Study, Roel Wieringa," *Reserach gate,* p. 7, 2017.

[14] J. C. S. d. P. L. Armin Eberlein, "Agile Requirements Definition: A View from Requirements Engineering," 2002.

[15] G. R. V. G.-Y. B. S. G. G. Junji ZhiShawn Shahnewaz, "Cost, benefits and quality of software development documentation: A systematic mapping," p. 24, 2014.

[16] s. S. E. H. J. J. H. J. Irit Hadar, "Less is More: Architecture Documentation for Agile Development," p. 4, 2013.

[17] V. MAHNIC, "Improving Software Development through Combination of Scrum and Kanban," p. 8, 2014.

[18] D. A. E. D. F. M. Frauke Paetsch, "Requirements Engineering and Agile Software Development," p. 6, 2003.

[19] A. Rüping, Agile documentation: A Pattern guide to producing lighweight documents for software projects, John Wiley & Sonds Ltd, 2003, p. 245.

[20] "Inforegister," 2018. [Online]. Available: https://www.inforegister.ee/10364097-HELMES-AS. [Accessed May 2018].

[21] A. K. Shenton, "Strategies for ensuring trustworthiness in qualitative research projects," p. 14, 20014.

[22] Signavio, "Business process managament tool," Signavio, 2009. [Online]. Available: https://www.signavio.com/.

Appendix 1 – The survey for the analysts during the interview

1. How long have you been working in this team?
2. Is your main client from public or private sector?
3. Describe your role in the team. If possible, list the main responsibilities
4. Do you consider your team as Agile?
5. What Agile methods do you consciously use in your team? How do you combine them?
6. How much have you heard about steps of requirements engineering? How many do you consciously use?
7. How long is your product backlog? (e.g how much work do you have planned for next iterations) Also how big are the average tasks?
8. Describe how knowledge is maintained in your team in the long run
9. Describe the process of documenting in your team (for example requirements or solutions) – how, when and what
10. Describe the format that you follow when documenting (for instance pure text, tables, models, diagrams, which diagrams, meeting notes)?
11. Describe the process of keeping the documentation up to date. What is the percentage of up-to-date-ness?
12. When do you document the solution in terms of the whole project (on the go, beginning, in the middle, on the end)?
13. What benefit attributes of documentation are the most important for your team (pick 2)
14. What quality attributes of documentation are the most important for your team (pick 2)

15. How much do you invest time in documentation during the week(including keeping it up to date)?

16. How much time does your team or other persons you know invest in reading the written documentation?

17. How much development time does your team invest in focusing on delivering new features to the client on weekly basis?(doesn’t contain maintenance, knowledge transfer, coaching)

18. How do you find the balance between delivering new features constantly and documentation?

19. What is the client’s opinion about documentation?

20. Anything else to add?

Appendix 2 – the link to the interviews

This link with is maintained by Google Drive and belongs to the author. It has been made accessible for everybody. In the folder, there are audio files together with transcribed documents and text files.

https://drive.google.com/drive/folders/13HOCGFo-ks83SmUaTQNiYb-9b6-xVhzH?usp=sharing

Appendix 3 – the process of documentation

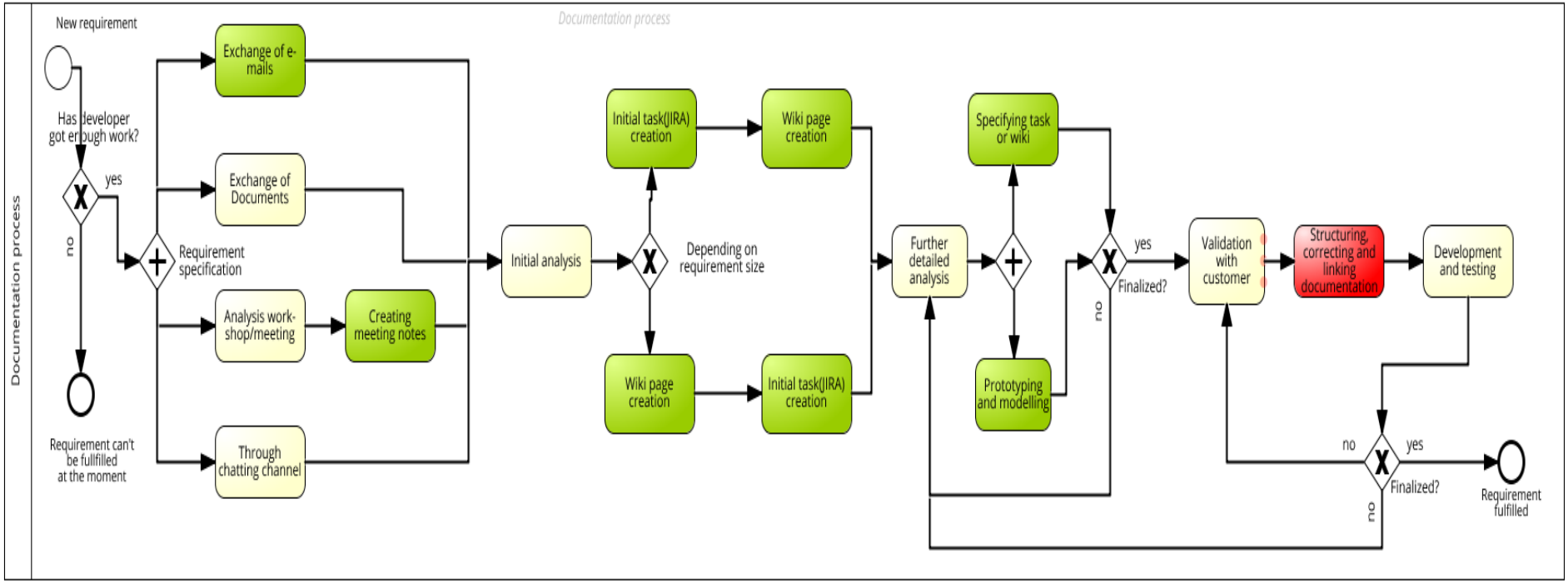

Figure 7. The process of documentation