INSIGHTS INTO WASTE IN AGILE SOFTWARE DEVELOPMENT

Michael Simader
Master of Science
Computer Science
December 2013
School of Computer Science
Reykjavík University

M.Sc. PROJECT REPORT

ISSN 1670-8539

Insights into Waste in Agile Software Development

by

Michael Simader

Project report submitted to the School of Computer Science
at Reykjavík University in partial fulfillment of
the requirements for the degree of
Master of Science in **Computer Science**

December 2013

Project Report Committee:

Marta Kristín Lárusdóttir, Supervisor
Assistant Professor, Reykjavik University

Asa Cajander
Lecturer, Uppsala University

Guðlaugur Stefán Egilsson
Software Developer and Founder, Sprettur

Insights into Waste in Agile Software Development

Michael Simader

December 2013

Abstract

Agile development processes are less prone to waste than traditional processes, e.g. waterfall model. Waste can be seen as non-value adding activities within the software development process, e.g. the loss of knowledge. Waste can remain invisible though, only the effects become apparent. The first step to address inefficiencies is to identify them. When IT-professionals are aware of waste, they may be able to address and eliminate it. Hence, the knowledge and understanding of waste is the cornerstone of diminishing it. IT professionals in Iceland were interviewed regarding waste in their organizations, all of which applied agile development processes. Even though there are efforts to reduce waste, these organizations are still confronted with problems attributed to miscommunication and the impediments by the external environment. The results of the interviews show the flaws within the delivery chain and provide software development organizations with important insights into waste. This can help to raise the awareness of waste in software development and eliminate non-value adding activities in general.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Business value is delivered relatively late in traditional development processes which raises the volatility of the project's budget and time estimates and lowers the adaptability of the software. Agile processes provide a solution to these problems by introducing incremental stages and focussing on delivering business value as fast as possible. As a result, agile software development has been a rising movement over the last years, especially in Iceland. Agile processes aim to alleviate disadvantages that come with traditional processes, e.g. the waterfall model. This might sound like a major improvement, yet there might be some flaws hidden in the correct implementation of agile processes and especially the adaptation to environmental factors. These environmental factors incorporate the interaction with customers for instance. As Highsmith (2009) stated interaction is more important than comprehensive documentation. It is crucial to balance the necessities and agility within the agile process and follow a steady improvement approach. Those flaws or inefficiencies might lead to hinder the ability to deliver value to the customer. Hence, activities occur that are non-value adding to the customer's business. This phenomenon is further described as waste.

Lean Software Development has developed along with the agile processes. Within Lean Software Development the elimination of waste is one of the most important principles. This set of principles can be utilized to support agile processes. Anderson (2012) describes that agile processes contain little waste and hence produce a better economic outcome than traditional processes. In order to accomplish the elimination of waste, it is necessary to be able to identify waste within the software development process. Hence, the knowledge and understanding of waste is the cornerstone of diminishing it. When IT-professionals are aware of the waste in their software processes, they may be able to address waste and eliminate it. This awareness is subject to the research presented in this thesis. Are IT-professionals aware of waste within the software development process? There are many commonalities between agile processes and Lean Software Development. This thesis mainly focuses on B2B software providers applying Scrum as development process. B2B software providers are inclined to have a closer relationship to the customers, this might cause problems to the same extent as it offers benefits, as the research will show.

Furthermore, this research should help IT-professionals to identify waste within their software processes. For this purpose qualitative research was conducted in form of interviews to find out what typical types of waste occur in Icelandic IT organizations. This does not include a quantifiable assessment of waste in general. The focus is merely on the perception of waste within the software development process. The work of Poppendieck and Poppendieck (2007) presents a mapping of waste types from Lean Management to software development. These Seven Wastes build the foundation for the qualitative interviews in order to categorize the types of waste within the IT organizations. These results can be used by IT professionals as a starting point to analyze waste within their development teams.

The following chapters cover the theoretical background that is necessary for the understanding of waste in software development. This is followed by the methodology and results. To complete with the results are discussed and a conclusion is given in the end.

Chapter 2

Background

The following chapter explains the theoretical background of the thesis. In the beginning the agile software development processes will be presented, Scrum in particular. Followed by an explanation of agile software development and the Lean Software Development principles. The Seven Wastes by Poppendieck and Poppendieck (2007) are discussed including the concept of value as their counterpart.

2.1 Agile Software Development

In February 2001 advocates of software development processes like eXtreme Programming, Scrum, Crystal and Feature Driven Development met to agree on the Agile Manifesto. It was remarkable that those 17 creators brought forward their common interests and philosophies to combine them into the Agile Manifesto rather than endorsing their own ideas coming from their expertise. The term **Agile Software Development** was born (Williams, 2012). The following table 2.1 depicts their ideas. Beck et al. (2001) state in addition to the manifesto the following: “That is, while there is value in the items on the right, we value the items on the left more.”

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

Table 2.1: Agile Manifesto as described in (Beck et al., 2001)

Individuals and interpersonal communication between stakeholders, also a rapid feedback based delivery of value, i.e. software, are in the center of attention. These methods are iterative and incremental, hence it is possible to adapt to a changing environment and circumstances (Beck et al., 2001). Highsmith (2009) criticizes the so called "plan-driven" methodologies for the missing communication and collaboration as follows: "Documentation is not a substitute for interaction." This is relevant for the results as they are presented in section 4.

2.1.1 SCRUM

Sutherland and Schwaber (1997) presented a paper on Scrum methodologies. Over the following years their experiences and industry best practices were merged into what is now known as Scrum. Scrum is an agile development process and belongs to the same family of management processes like eXtreme Programming or Crystal. All of which are characterized by the agile manifesto as shown in section 2.1. In this line of agile methods Scrum is a widespread and the most commonly used process (Leffingwell, 2011). As the work of Larusdottir (2009) has shown, 37% of the respondents in her study apply Scrum as development method, whereas 44% use their own process the rest 19% account for XP or traditional methods, e.g. the waterfall model.

Sutherland and Schwaber (1997) also defined the roles as follows, *Product Owner*, *Scrum Master*, *Scrum Team*. The *Scrum Master* coaches the self-organizing and self-accountable team and is responsible for the compliance to the process. He handles the Scrum Team's problems that impede their work and removes obstacles. The *Product*

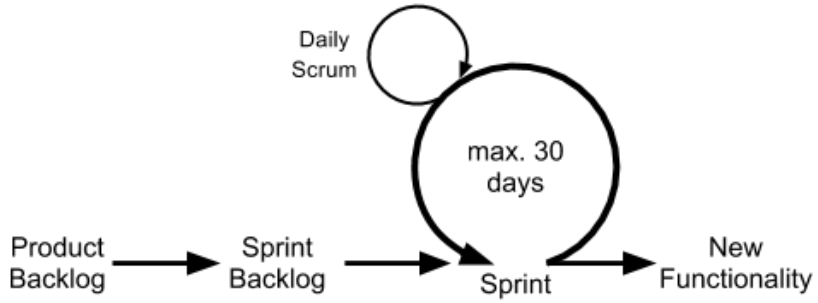

Figure 2.1: Scrum Process (Schwaber, 2004)

Owner can be seen as the customer proxy and is responsible to represent the needs and ideas of other stakeholders and the business. He also defines and maintains the user stories, which are collected in the *Product Backlog*, as functional and non-functional requirements to the software. The *Scrum Team* is a self-organizing and self-accountable team and works on and develops independently the software.

A development cycle is called *Sprint* and the duration should be between 2 and 4 weeks. Figure 2.1 depicts an overview of the process of Scrum. The user stories are prioritized by the *Product Owner* and the *Scrum Team* estimates and chooses user stories which they commit on to develop in the following sprint. The *Scrum Team* collects feedback from the customers and reacts to it accordingly in the following sprints. Scrum is an agile approach and adheres to the principles of the Agile Manifesto, as can be seen in table 2.2.

Principles in Agile Manifesto	**Realization in Scrum**
Individuals and Interactions	Team manages Sprint Backlog Team is self-organizing and self-accountable Daily Scrums
Working Software	New functionality each Sprint
Customer Collaboration	Product Owner Sprint Planning Sprint Review

Principles in Agile Manifesto	**Realization in Scrum**
Responding to Change	Daily Scrums Sprint Planning Sprint Retrospective

Table 2.2: Agile Manifesto and the Realization in Scrum

2.2 Lean Software Development

Lean Management is a holistic approach towards the objective to provide the right product in the correct amount at the right place and time in the right quality. Lean Software Development is not a process like Scrum but can be seen as as a set of principles extending the ideas of agile development (Kniberg, 2010). The following section summarizes the seven principles in Lean Software Development. The first and most important principle is the subject of this thesis. It has been stated by Ohno (1988), that the following principles are necessary in order to satisfy the first principle, Eliminate Waste.

2.2.1 Principle 1: Eliminate Waste

According to Ohno (1988) the highest goal to pursue is “the absolute elimination of waste”. This can be achieved by removing non-value adding waste in the timeline of starting with receiving an order until ending with collecting the cash. The phases of value creation lie in between those two points in time, e.g. requirements engineering and software development. It is necessary to develop an understanding of what customers value in the product. This value might change over time, as the customers do not really know what they want. Even after the software had been distributed customers might change their perspective of value. The next step is to identify non-value adding waste. Anything that does

not contribute to a customer's value can be considered as waste. Poppendieck and Poppendieck (2007, p.23) developed a mapping of types of waste from the original Toyota Production System to software development, as can be seen in the section 2.3.

2.2.2 Principle 2: Build Quality In

The avoidance of defects in the product is more important than testing the quality into the product at later stages. Hence, the focus is not on managing defects in a tracking system, but to prevent them in the very beginning. Defects managed in a queue within a tracking system can be considered as a waste of the type Partially Done Work, compare with section 2.3. (Poppendieck & Poppendieck, 2007, p.25)

Shingo (1982) argued that there are two types of inspection: inspection to prevent defects and inspection after defects occurred. If a defect is found, the production is stopped, and the cause for the defect should be detected and removed immediately. This requires a highly disciplined organization. (Poppendieck & Poppendieck, 2007)

2.2.3 Principle 3: Create Knowledge

Software development is a constant process of knowledge creation. Even if knowledge exists prior to the actual coding, in form of requirements, the detailed design and architecture will come in further stages of the development process. Early designs can not take the complexity into account, as it is revealed in the implementation process. Furthermore, as aforementioned, customer's change their perspective of what they want over time, a fixed and inflexible design can hinder the process of creation and is not open for feedback. (Poppendieck & Poppendieck, 2007, p.29)

2.2.4 Principle 4: Defer Commitment

The first goal should be to make situations based on decisions reversible, so they can be easily changed afterwards. This does not have to be applied on every decision, but the software system should be open for different options at points changes are likely to occur. If a decision is irreversible, the decision maker should wait until the latest possible moment. At this moment most of the information is present and it is more likely to make the correct decision. Although team-leaders tend to manage uncertainty in their early occurrence and estimate risks. Leaving options open and experimenting with them can help with gathering information and postpone irreversible decisions to the latest possible point of time. (Poppendieck & Poppendieck, 2007, p.32)

2.2.5 Principle 5: Deliver Fast

One of the main principles in agile methodologies is to deliver features with the highest business value first. This has a positive effect on the customer satisfaction, as the customer can start using the software in early stages of the project. With each iteration new business value is delivered to the customer. As there are also changes involved in this process, the development team must have good reflexes and be disciplined in order to keep the level of quality on the desired level. This is supported by a continuously improving environment and engaged and skilled people. Quality in general can be built in in two ways. Firstly, following a slow and careful approach, which might have negative effects on the customer excitement and changes may impede with the process even more, which leads to more waste. Secondly, a fast and reflexive approach with continuous feedback from the customer. (Poppendieck & Poppendieck, 2007, p.34)

2.2.6 Principle 6: Respect People

In agile methodologies the awareness of people who add value to the product must be high and those people need to be respected by empowering them to develop their own practices and improvements to the process. This leads to a higher level of problem-solving and reflection skills. (Leffingwell, 2011, p.23)

2.2.7 Principle 7: Optimize the Whole

As stated in Principle 1: Eliminate Waste the main goal is to optimize the whole value stream within an organization. It is not enough or might be even counterproductive to concentrate on single aspect of the value stream. This needs to be supported by the whole organization and the managers need to be able to defer responsibilities to lower levels (Poppendieck & Poppendieck, 2007, p.38). Auxiliary tools can be used to give an overview of the situation, e.g. Kanban-board.

2.2.8 Kanban Overview

Kanban is a lean approach to agile software development. Kanban is a tool for visual management and literally means “visual card”. The following listing shows how Kanban works (Kniberg, 2010):

- **Visualize the Workflow** Split the work into smaller chunks, write each item on a card and attach them to a wall
- **Limit Work in Progress (WIP)** assign explicit limits to which how many items may be in progress at each workflow step
- **Measure the lead time** Average time to complete one item (cycle-time), optimize the process to make the lead time as predictable and small as possible

Kanban does not define particular roles or meetings, but can be added as needed, e.g. Daily Meetings as they occur in Scrum are commonly used. Kanban describes the Kanban-Board as a visiual instrument as the only artifact. Again, further artifacts can be added to the process. However, Kanban is a light-weight and agile approach and meetings, roles and artifacts should only be carefully and justifiably added to each individual process (Kniberg, 2010). The highest goal is to eliminate waste, not to introduce additional waste.

2.3 The Seven Wastes

Shingo (1982) has studied lean manufacturing and identified seven wastes of manufacturing. Poppendieck and Poppendieck (2007) mapped these seven wastes to software development equivalences, as table 2.3 shows.

Manufacturing	**Software Development**
In-Process Inventory	Partially Done Work
Over-Production	Extra Features
Extra Processing	Relearning
Transportation	Handoffs
Motion	Task Switching
Waiting	Delays
Defects	Defects

Table 2.3: The seven wastes mapped to software development (Poppendieck & Poppendieck, 2007, p.74)

Partially Done Work is present when artifacts in the development process are impeded in the process of start of work to integrated, tested, documented and deployable. Partially done work can be diminished by dividing work into smaller chunks or iterations. (Poppendieck & Poppendieck, 2007, p.74) Middleton, Flaxel, and Cookson (2005) describe that bottlenecks delay the product from being deployed, therefore a continuous flow should be pursued. A high amount of requirements put into the system is mentioned

as a typical example. Further examples for partially done work are Uncoded Documentation, Unsynchronized Code, Untested Code, Undocumented Code, Undeployed Code. (Poppendieck & Poppendieck, 2007, p.74)

Extra Features are similar to over-production, which is deemed by Ohno (1988) as the worst of the seven wastes. The feature should not be developed, if there is no clear value for the customer, or the feature does not support the accomplishment of the customer's current job. (Poppendieck & Poppendieck, 2007, p.75)

Relearning stands for rediscovering forgotten knowledge. Therefore it is essential to create and preserve knowledge as part of a learning process. Further, it is crucial to utilize existing knowledge and experiences from employees. (Poppendieck & Poppendieck, 2007, p.76)

Handoffs happen when knowledge is transferred from one colleague to another. With every single handoff tacit knowledge gets lost, because it is very difficult to make tacit knowledge explicit. Poppendieck and Poppendieck (2007) state that only 6 % of the original knowledge is left after 4 handoffs. Therefore it is necessary to reduce handoffs in order to reduce waste. (Poppendieck & Poppendieck, 2007, p.77)

Task Switching requires knowledge workers to reset their mind after each switch. This resetting is time consuming and therefore waste. It is considered, that working on tasks in sequence compared to in parallel is more time efficient. As a result the number of tasks ought to be kept to a minimum in order to reduce task switches. (Poppendieck & Poppendieck, 2007)

Delays occur in many different situations. One of the most important types of delay is waiting for people in different areas. Developers make critical decision about every 15

minutes. These decisions can only be made, if the required information is present. Co-located teams with short iterations and regular feedback can provide developers with the information they need to make decisions without delay. Hence, knowledge needs to be available when and where it is needed. (Poppendieck & Poppendieck, 2007)

Defects within the software cause numerous problems and may lead to customer dissatisfaction. The target should be to deploy or deliver the product with the lowest possible defect rate. Mistake-proofing tests and the discovery of defects in early stages should be in focus of the whole development process. (Poppendieck & Poppendieck, 2007)

2.4 Value

Waste is non-added value within the defined timeline. Hence, it is necessary to study the value when studying waste. The view on value varies from company to company and also if product development or project management is exercised. Within product development oriented organization value might be market-share or profitability, whereas in project management organizations value could be understood as the ability to meet costs, schedule and scope commitment constraints. (Poppendieck & Poppendieck, 2007, p. 61)

Software applications can be disassembled into units of marketable value, into so called Minimum Marketable Features (MMF). Every single MMF can be sold to the market independently and generates value for the customer. Apparently a software product has value to the customer. However this value is not experienced as a whole, but as the series of valuable features. Utility software might only consist of 1 MMF and more comprehensive software of many more, still it is of value for the customer (Denne & Cleland-Huang, 2004). This results in different value expectations for different customers, which need to be balanced and considered for the MMF delivery.

Chapter 3

Methodology

This chapter comprises the description of the applied methodology. In order to understand the perceived state of waste in Icelandic software development teams interviews were conducted. The structure of the interviews, informants and analysis are described in the following sections as well as the limitations of the methodology.

3.1 The Interviews

The qualitative study included 10 interviews, all following the same structure asking for the experience and context of the informants within the organization. Questions to the informants were related to the seven wastes defined in the section 2.3 and were aiming on the perception of waste in the informant's organization. The introductory part was focussing on the experience of the informants and the applied processes to get an opinion about the context. Whereas the main part was particularly aimed on the seven wastes and how they are perceived. Followed by questions about a continuous improvement process and other common problems in the development process. The table 3.2 shows the template of the interview questions.

The interviews were conducted mostly on site of the informants' organizations in a quiet and undisturbed environment. Two of the interviews were conducted in a meeting room at Reykjavik University. All of them lasted for about 45 minutes. The interviews were all conducted in English and the conductor also took notes and recorded the interviews. There were no technical problems encountered during the interviews. All recorded interviews were transcribed verbatim, with slight modifications to make the text more readable, when filler words or phrases disrupted the structure of a sentence. Informants are addressed as males despite their actual gender when describing their comments.

3.1.1 Companies

The companies were chosen through personal contacts supported by data by the Statistics Iceland office (*Statistics Iceland office*, 2013) and the Icelandic chamber of commerce (*Icelandic Chamber of Commerce*, 2013). The focus was on businesses either providing bespoke or off-the-shelf software, hence only companies selling B2B software were considered in this research. Those companies were equally balanced and 5 were chosen from each category. The categories are described in section 3.1.2. This helps to analyze a possible difference in the meaning of waste, because the costumers base is different and hence the communication with them.

All the chosen companies exert agile management methodologies, and Scrum in particular. Some of the companies have adapted the Scrum process to their own needs and/or also apply Lean Software Development principles.

3.1.2 Informants

The informants were found through personal contacts and recommendations within the chosen companies. All of which have several years of experience in the application of agile methodologies, especially in Scrum, and worked also as members of the development teams. The roles of the informants vary from company to company. The informants held the roles depicted in table 3.1 within their organizations. The table also shows the allocation of these roles regarding the distinguished groups and the hierarchy levels within the organization of each role (1... Executive Level, 2... Upper Management, 3... Development Team Level).

Level	**Role**	**Bespoke**	**Off-The-Shelf**
1	Chief Technology Officer or Director of Development	1	2
2	Head of Development Department	2	2
3	Scrum Master or Product Owner	2	1

Table 3.1: Overview of the informants' roles

The interviewees did not prepare for the interview in advance, because they were only asked for their experiences they had made within the prevalent environment. Any preparation might have led to a biased opinion and it can be assumed that the preoccupation might have lead to results that do not reflect how waste is perceived but defined. Furthermore, the informants are categorized in two different groups, depending on the type of software that is developed in the informants' organization. The 2 groups are as follows BESPOKE-Software and OFF-THE-SHELF-Software. This categorization is made because Poppendieck and Poppendieck (2007) point out the different focus on value for these groups, as discussed in section 2.4.

3.1.3 Interview Questions

Table 3.2 shows the interview questions which are linked to either the seven wastes or the context of informant's organization. These questions can be understood as a guideline in order to get as much information as possible and give the informant a high degree of independence within the boundaries. Furthermore, the interviewees did not need to prepare for the interview, because the participants were only asked about their experience and perception of waste within the prevalent context.

	#	Question
Informant Context	1.1	Describe the field of operation of your company.
	1.2	What kind of software development process do you use? When did you start with this approach?
	1.3	What is your experience with other approaches?
	1.4	What is the structure of the teams working in Software Development?
	1.5	How do you define the term waste in your own words?
Waste Context	2.1	How do you describe Partially Done Work in your organization?
	2.2	Describe the process of requirements engineering. Do you have the impression you would implement more feature than demanded?
	2.3	How are requirements managed in collaboration with the customer?
	2.4	Do you think knowledge gets lost in the progress of a project, which makes relearning necessary?
	2.5	Describe handoffs in your development process.
	2.6	How are tasks assigned?
	2.7	How many tasks has a developer to process at a time in average?
	2.8	What are the main reasons for delays in the development process?

	#	Question
misc.	2.9	How do you manage defects in your projects?
	2.10	How would you describe the process of continuous improvement in your organization?

Table 3.2: List of Interview Questions

3.2 Qualitative Data Analysis

The data from the interviews were compiled, analyzed and assigned to categories eventually. Interpretative Phenomenological Analysis as described in (Silverman & Rapley, 2011, p.274) was applied to identify and generate themes and sub-themes as depicted in table 3.3. This was an iterative approach and the themes were refined with every single transcript of the interviews, which results in the final themes and sub-themes. Notes and the assigned themes were directly put down on the printed transcripts. First themes were generated, which lead to an initial list of themes. The themes in this list were clustered, which resulted in a list of themes of connected areas. As the last step these themes were organized in a table consisting of themes and sub-themes. These steps were repeated for the following transcripts and the table was refined to the resulting table 3.3.

Theme	Sub-Themes
Definition of Waste	-
Partially Done Work	Inventory Defect Backlog
Extra Features	Customer Involvement Unused Features
Relearning	Loss of Knowledge Team Communication

Theme	**Sub-Themes**
Handoffs	-
Task Switching	Task Assignment Number of Tasks
Delays	-
Defects	-
Continuous Improvement	Process Metrics

Table 3.3: Themes and Sub-Themes according to Interpretative phenomenological analysis (Silverman & Rapley, 2011)

The interviews will be analyzed according to the labels and the results will be presented. Furthermore differences between the two identified groups will be discussed. In the following the informants are referred to the following coding:

- BESP-#, belonging to the *BESPOKE-Software* group
- PROD-#, belonging to the *OFF-THE-SHELF-Software* group

3.3 Limitations

The limitation of the thesis pertains to the chosen informants mainly. There were 10 interviewees from Icelandic companies chosen as presented in section 3.1.2. Those participants are part of organizations that develop B2B software only. It was assumed that business customers from the defined groups are more homogenous, hence comparative results of these groups are more significant. Furthermore, the assessment of waste pertains to the perception of the interviewees only. There was no quantitative analysis towards the seven types of waste conducted.

Five informants for each group may be a low number, this is attributed to the fact, that the author of this thesis conducted the research individually, which also resulted in time constraints. Nevertheless, the consistent results among the groups show, that this did not influence the quality of the data substantially. Furthermore, it was difficult to estimate how many companies operate in Iceland that offer B2B services, as there were no reliable sources present. The following table shows the number of ICT enterprises and people employed in ICT sector in Iceland, these numbers include all ICT providers, also ICT wholesaler or consulting firms, but not banks that have software development departments incorporated. Assuming that half of these companies would provide software development services and 35% of these would apply agile processes, as the work of Larusdottir (2009) has shown. Additionally, 70% of these would develop B2B software.

Year	**ICT sector total**		
	number of enterprises	assumed relevant population	employees
2005	422	52	6,145
2006	456	56	6,394
2007	459	56	6,349
2008	433	53	6,120

Table 3.4: Number of ICT enterprises and employees in ICT in Iceland (*Statistics Iceland office*, 2013)

Ten representatives out of nine companies were interviewed, this is equal to 20% of the assumed and restricted population of companies. These numbers were pessimistically appraised. The size of the software development divisions ranged from 5 to 80 software developers and 15 in average within the companies.

Chapter 4

Results of the Interviews

In the following chapter the results of the interviews are presented. The results are structured according to the identified themes in the analysis process. The first section describes the informants' definition of waste in software development. Furthermore, the results of common problems within software development teams and customers are depicted. To conclude with the results pertaining to the two identified groups are compared.

4.1 The informants' definition of the term waste in software development

The following table 4.1 shows the results of the informants' definition of the term waste in software development. The informants were not prepared for this question, in this case they responded with their personal experience and not with a biased opinion of waste in software development.

Informant	Definition
BESP-1	Waste is something that you redo, which should have been done in the first place. It can be a programmer, the project manager, the manager, whatever. It is something you do twice, when you could have done it once.
BESP-2	Waste is developing the same requirements many times. The problem is not the correct understanding of the requirements or business, or the customers would change their minds. Waste is also when systems are not designed well, and the system evolves over time, which raises the complexity. A lot of waste can be found in refactoring, although the system maybe does not have complex functionality, but bad design makes it difficult to train developers. If quality is not assured in the early stage of the process, defects might end up more frequently in the released software. Misunderstanding of requirements with the customer, when the user was not involved.
BESP-3	Waste is anything that you have done, that would not have changed anything about the value that you are delivering or the ability to deliver that value. Anything that does not contribute to the value is waste.
BESP-4	Waste is work that does not contribute to the value.
BESP-5	Waste is, when you are working on a badly defined project and the project is delayed. The project cannot be put live, because of outstanding decisions that need to be made. When projects cannot be finished because higher prioritized projects interrupt, which leads to putting the first project on hold.
PROD-1	Waste is something unnecessary, if you do something or make something, that is not used or means extra effort.

Informant	Definition
PROD-2	Waste is noise created by the external environment. When developers are distracted or have direct contact with the customer and the customer asks for new features directly. This feature is developed although it did not go through a thinking process, and it might even not be needed anymore.
PROD-3	Waste is when you are doing something that is not needed. Or you are doing something that is not delivered. When you are doing something that nobody needs or nobody wants or you are not shipping it.
PROD-4	The major waste in software development is software that has already been written, that is too specialized and customized, so you cannot use it for other customers and it does not fit in the structure of the framework. Waste are also solutions of low quality, that are not tested enough and do not fulfill the requirements very well.
PROD-5	Waste is, if you develop a feature or function that you end up with throwing away, you may have even known it beforehand, because you are following the old-style waterfall model. You know after a while that it is not going to be of any use, but it is in the contract somewhere in the business requirements document. Developing a feature that will be needed in the future and you start working on this feature although it will be needed in a few months, so you will put it on the shelf. There is a lot of waste that goes on, if you are not process driven and you have to switch contexts a lot.

Table 4.1: Results of the informants' definition of the term waste in software development

The informants commented on waste in many different ways. BESP-4 had a very clear definition of waste,“Waste is work that does not contribute to the value.”, whereas the others were referring to more specific types of wastes, e.g. PROD-4 mentioned Partially Done Work when he said,“The major waste in software development is software that has already been written, ..., but you cannot use it for other customers.”. This also includes activities that have to be done repeatedly, because there was not a common understanding about the requirements with the customer. It also seems that the complexity of the systems is a problem in the development process, as BESP-2 and BESP-5 mentioned it very clearly. Furthermore, noise and distractions from the outer environment was considered to be waste for PROD-2.

The table 4.2 shows an analysis of the interviews against value, the counterpart to waste. It was counted how often the informants refer to value during the interview.

Informant	**Number of references to value**
BESP-1, PROD-1, PROD-2, PROD-5	0
BESP-2	1 ‘clear value adding features’
BESP-3	17 ‘delivering value’, ‘contribute to the value’, ‘business value’, ‘prioritizing value’, ‘item of value’, ‘achieve the value’, ‘responsible for the value’
BESP-4	1 ‘Waste is work that does not contribute to the value’
BESP-5	2 ‘value on the market’

Informant	Number of references to value
PROD-3	2 'Value that we are getting','Value goes up for the customer'
PROD-4	1 'Value is sellable functionality'

Table 4.2: Frequency of references to value during the interviews

The informants did not mention the customer's value very often. Only *BESP-3* was emphasizing on value throughout the interview. He also highlighted the necessity of being as fast as possible and involving the customer is very important, "We are so focussed on the value of delivering fast and getting response." Delivering fast was also very important to *PROD-1*,"Have a short turnaround time for everything. Move fast!", also he saw the need of releasing in shorter cycles,"We are constantly working on speed, quality, and reducing the release cycles".

4.2 Occurrence of Partially Done Work

The main results on the Occurrence of Partially Done Work include:

(a) Defect backlog is not recognized as Partially Done Work.

(b) Unfinished features do not generate value for the vendor and the customer.

(c) Communication over heavy-weight documentation.

Nine out of 10 informants reported that unfinished features or non-fulfillment of requirements would be the most common problem when it comes to partially done work. The documentation of the process or products seems to be not a common problem. The

informants were referring to the communication structure within the team as being supportive for neglecting heavy weight and completed documentation. *PROD-2* for example explains the lack of documentation as follows, “Within the team everyone is telling everyone about everything.”. *PROD-4* explains the lack of documentation with “We have daily stand-ups and show and tell sessions.”.

The informants do not have problems with unfinished, untested or undocumented code. This is attributed to the used process in development. The developers pick a task and finish it all the way until testing. *BESP-2*, *BESP-3*, *BESP-5*, *PROD-1*, *PROD-3*, and *PROD-5* have testing separated from the development, the rest of the informants reported, that the developers also test the software. All of the informants stated, they would use KANBAN-like boards or status walls for better visibility of the status of the software artifacts, e.g. *BESP-2* said, “We do visual management, so we have every ticket up on the wall.”.

Although the informants do not recognize a defect backlog as partially done work, but 8 out of 10 maintain an inventory of defects, except for *BESP-3* and he said,“It’s an absolute waste to collect huge backlogs of defects, that you review every 2 months and it’s only the top 10% that’s going to get ever implemented.”. This is considered to be waste in the terms of partially done work, as defined in section 2.2.2. *BESP-3* explained further, “We have a zero bug policy, although this is utopian, but we try at least.”.

4.3 Occurrence of Extra Features

In the following the results of the waste type *Extra Features* are presented. First of all the customer involvement in the process will be discussed, followed by the occurrences of extra features and the efforts of the informants to streamline the products. The assumption is, the closer the customer is involved within the process the less extra features occur. This is attributed the higher degree of communication, which leads to a lower level of misconception of requirements and hence a lower amount of extra features.

4.3.1 Customer Involvement

The main results on Customer Involvement include:

(a) Close relationship and customer involvement is very important to reduce misunderstandings.

(b) Customer needs to be educated to collaborate in an agile environment.

(c) FAQs are a helpful way to handle a high amount of users with a small team.

The following results show the customer involvement in the process of software development. This exceeds the regular requirements engineering process and shows how the informants stay in touch with the customer throughout the development process. The term customer refers to different stakeholders on the customer side. This might also include users.

Most of the informants shared a common desire for a high degree of customer involvement. A close relationship can prevent from misunderstandings about the requirements and can even make training of the user irrelevant. *BESP-2* reported about a loose relationship with the customer, which led to misunderstandings and a higher amount of later

improvements, once the software had been delivered. Most of the interviewees prefer direct communication via phone over emails, for the reason to prevent misunderstandings again and it is “more convenient to just pick up the phone”, so *BESP-3*.

The customers need to be familiarized with this close involvement as *BESP-1*, *BESP-3* and *BESP-4* reported. *BESP-3* explained this as follows, “The customer needs to fit the agile process in their environment. They must learn it and it is hard in the beginning.” *BESP-1* also argued that, “some customers want to stay with the plain and old-fashioned waterfall model, and this is reflected in the nature of the tenders.”. The need for educating customers was high for *BESP-4*, because they were still reluctant to give feedback, “We ask for feedback, but the answer comes a lot later.”. *BESP-4* also spoke about a decrease of service level experienced by the customer, because they shielded the developers from the customer in order to reduce distractions, “but in fact the service level is actually increasing”. Furthermore, *PROD-5* said, “Interesting side effects also are, after developing a big piece the training was basically nonexistent.”, because of the high customer involvement.

PROD-3 provides FAQs for the customers, to support them with installation and configuration of the product and to reduce the amount of support requests to the help-desk. *PROD-5* complained about the process and the interaction with the customer was less agile due to the acquisition by a large corporation, “We are less agile today than before the acquisition.”. He reckoned, it was not possible anymore to involve the customers as much, although when “the customer invests the time, then this is really beneficial when it comes to waste”.

There is are differences noticeable between the two interviewed groups. The degree of customer involvement is much higher in the *BESPOKE-Software* group than in the *OFF-THE-SHELF-Software* group. This can be explained with the nature of the customers. On

the one hand *BESPOKE* customers actively request the software product on the other hand the *OFF-THE-SHELF-Software* group tries to advertise the product to a broader audience of customers, without changing and customizing the product too much.

It comes all down to the delivering the right value to the customer. *OFF-THE-SHELF-Software* providers need to think about value differently as they need to balance the delivered value with the individual expectations of the customers, which is described in section 2.4.

4.3.2 Extra Features

The main results on Extra Features include:

(a) Extra features do occur, but there are efforts to streamline the solution.

(b) Extra features can be an opportunity to learn.

(c) It is better to exceed the expectations, than to fail to meet the basic requirements.

Extra features occur in the informants' organizations. Some interviewees stated, they knew of this issue, whereas others claimed to exactly develop what is demanded by the customers. There are different reasons and approaches to face this problem though. *BESP-1* did not see a problem with extra features, because even though they cannot charge this extra effort, "but in the long run, we do not lose money with it, because of repeat business" and they want to live up to the expectations and even exceed them. Also *PROD-1* appraised this issue similarly, "We learn from it and we profit from the knowledge we gain from it.". Reasons for extra features can often be found in miscommunication, and hence not knowing exactly what the customer demands. *BESP-2* has the problem of a "non-mutual understanding of what the users and what the business customers are actually asking for". In order to "keep the solution user friendly" *PROD-1* stated they removed

features and streamlined the solution. The same happens in *BESP-3's* organization, because "it's a work system, there are people in there and there is no value having this feature in there" when it's not used.

Only *BESP-3* stated to take the effort to analyze which features are valued by the user in the productive system. They systematically remove them in succession, if there is no use for a particular functionality. For the rest, no real efforts were reported to estimate the value of a feature for the customer.

4.4 Relearning

In the following the results of the waste type *Relearning* are presented. First of all the loss of knowledge will be discussed, followed by the structure of the team communication. The assumption is, the better communication works the less knowledge is lost in the process of software development. Even though knowledge is not tangibly available, it will be explicit due to intensive interaction among the team members.

4.4.1 Loss of Knowledge

The main results on Loss of Knowledge include:

(a) Preservation of knowledge mainly supported by the open communication structure.

(b) Tools are used to support the communication, such as wiki-pages and issue tracking systems.

None of the informants responded, that they would have a problem with the loss of knowledge within the development process. This is attributed to the communication within the teams, as the following section 4.4.2 shows. *PROD-2* stated, that about half of

the development team left the company and this did not cause troubles, because “the application is tested, everything is tested and we use a high level language, so it’s fairly easy to navigate through and also because everyone is telling everyone about everything.”

The utilization of heavy-weight documentation is mainly used for contractual reasons, because it is requested by the customer. As *PROD-5* stated their customers would demand detailed documentation about the project, furthermore “The process is too partitioned, hence, documentation is needed, because there are so many people involved.” in the whole project development life-cycle. Some of the respondents maintain wiki pages and issue tracking systems for general documentation purposes, but they also emphasized, that most of the knowledge is share in an open communication process.

4.4.2 Team Communication

The main results on Team Communication include:

(a) Knowledge is distributed by respectful and open-minded face-to-face communication.

(b) Trade-off between individuals with expert knowledge and commonly shared knowledge.

All of the respondents utilize daily stand-up meetings and encourage team members to communicate face-to-face to share knowledge within the team. Most of the informants facilitate KANBAN walls, e.g. *BESP-2* explains the advantages of this tool as follows, “So we have all the tickets up on the wall, and this makes all the tasks visible for everyone.”. In general the interviewees emphasize the open and respectful communication within the teams. *BESP-4* even reports about a social contract that is concluded among the team members as follows “Everyone is open minded about asking questions and giving feedback.”. As teams in Scrum are self-organizing, *BESP-4* also talks about, that the

teams decide what is important and needs to be written down and what not. Also *BESP-4* highlights the level of respect and cooperativeness within the software development team when it comes to integrating new team members, “Everybody would be so helpful, it’s really good to pick up speed with a new team member”.

Some of the interviewees reported about rotation within the teams to distribute knowledge, like *PROD-4* stated, “it is helpful to be at least knowledgable of each part of the code” and *BESP-3* is of the same opinion. Whereas *BESP-5* and *PROD-5* think that experts are important. *PROD-5* argues with the high complexity within the product, that make experts necessary. Also he admits, that “a single point of failure” is adverse and it is required to find the right balance. For *BESP-4* experts might be distracted by other team-members as they are a valuable source of information and the single contact point. *BESP-3* declined the evolvement of experts within the team and reported that they tried to share knowledge as much as possible, by using e.g. pair-programming or team-member rotation.

4.5 Handoffs

The main results on Handoffs include:

(a) Loss of knowledge during handoffs can be minimized by communication structure.

(b) Loss of knowledge happens at the interfaces between customer and development.

The results show that there is a certain pattern regarding handoffs identifiable, as shown in figure 4.1. These 3 steps occur in all of the informants’ organizations. The single stages are shaped differently within the organizations. Although the development process is always characterized by ‘1 developer works on one task, for bigger tasks the work is divided into smaller tasks’.

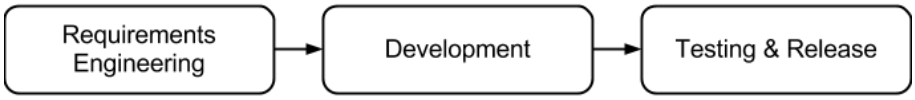

Figure 4.1: Basic Handoffs

The requirements are elicited in collaboration with the customer for the *BESPOKE-Software* group. The *OFF-THE-SHELF-Software* group maintains wish-lists, which are compiled from requests by the customers. In general the user stories are developed by the developers themselves and also tested.

BESP-1 saw the problem that knowledge gets lost when the requirements in form of user stories are handed to the developers, “There is probably a loss of information ongoing.”. He also debilitated this effect, “because we sit in a really open space, so there are a lot of discussions going on.”. *PROD-5* described handoffs as problematic in the requirements engineering process and the delivery process, because the development team is embedded in a large corporation and the distance to both ends to the customer is rather high. “The customer is fairly isolated from the development.”, additionally “For me I think this is too partitioned, too waterfall. Especially, a lot of information is lost, when you have a totally separated team talking to the customer. So what is developed may have lost some context.”. *BESP-3* tries to tackle this problem with the following solution,“Continuous delivery. It’s a thinking of minimizing these handoffs and the costs of handoffs. So we have a deployment pipe all the way into operation.”. He also reported that the same team working all the way from kick-off to the operation eliminated a lot of waste.

4.6 Task Switching

In the following the results of task switching are presented. First of all the process of task assignment is shown, followed by how many tasks a developer has to handle at

a time. This should depict the problem of context switching, which is also described in section 2.3.

4.6.1 Task Assignment

The main results on Task Switching include:

(a) Ideally, developers work on 1 task at a time, higher prioritized tasks may interrupt.

(b) Team members pull their tasks from a task-pool.

Tasks are not assigned to team members, as all of the informants report that they apply Scrum, an adapted Scrum process or Kanban. The teams are self-organizing and render all of the following methods independently: Estimation, Planning, and picking or choosing the tasks individually. This is compliant to the agile principles as described in section 2.1. *BESP-2* described the situation in this fashion, “The developer picks the task and then puts a sticky note on the board.”. *BESP-3* had the same opinion and added the following,“It’s the agile and lean principle of pull system over push system, so the team pulls the tasks”. *BESP-4* came to the point with “This happens in the daily stand-ups, team members are self-organized and choose the tasks. We do not assign tasks. You commit to a task.”. *PROD-2* exclaimed also, what all the other informants have in common, that tasks are chosen accordingly to the priority, “If you are a developer and you see the accepted tickets, there are 3 highly prioritized, so you pick 1 of those 3.”.

4.6.2 Number of Tasks

The main results on Number of Tasks include:

(a) Ideally, 1 task at a time. In reality, not always realizable.

(b) Defect reports interrupt the development process and lead to context-switching.

The informants agreed to limit the number of tasks to just 1 task at a time, but this is not always possible due to many different reasons. Incoming defect reports can interrupt the development process, because defects need to be given a higher priority for some of the informants. *BESP-4* explained this as follows, “Developers should work on 1 task at a time, but on the daily stand-ups they can choose more, because the planning period is 24 hours and they can finish more tasks within that period. Except if there are some defects coming in, that need attention”, then the developer has to switch the task. *PROD-3* and *PROD-4* also explained the switch to another task, when a higher prioritized defect is reported. In this organization a special role was introduced that is assigned to a different team member every sprint cycle, which only handles defect reports. *PROD-1* has a similar opinion, but also urgent customization requests from customers could interrupt a current task. *PROD-2* saw this problem more strictly and argues, “If a developer needs to work on another ticket, the first has to be put on hold, then you can see how the hold queue is piling up. I think it’s cumbersome to have many tickets or working on many things at the same time”. *BESP-3* reported that developers only work on 1 task at a time, but the method of pairing is used, so there might be more tickets assigned to one developer than he actually works on. *BESP-5* pictured the situation in his organization as follows, “Ideally 1 task, in reality things tend to get a lot worse than that. So context switching is a problem, especially for certain team members that are very knowledgable.”.

4.7 Delays

The main results on Delays include:

(a) Waiting for decisions from outer environment is the main cause for delays.

(b) Late detection of defects within the development process leads to delays.

(c) High complexity delays the development process.

The informants reported different reasons for delays, but there are two main themes identifiable. First of all there is a blocking in the development process due to missing actions by an outer stakeholder. *BESP-1* explains it as follows, “We are not synchronized enough with them. They have to do something and then we have to wait and cannot move and then they have something done and then we have to go on. I think within the department things run quite smoothly.”. Miscommunication and a lack of clear responsibilities can be seen as a reason for that. *BESP-3* had a concise answer to this in order to prevent this from happening, “One of our tag lines is - Responsibility all the way. We are responsible for the value and this story has no value until this blocking is resolved, we have to finish it. So we are offering our help, call them, and make sure things happen.”.

Poor software design and the resulting complexity delays the development process, at least for *BESP-2*. It takes a while until the developer understands the problem and structure of the software. This leads to the next common problem, fixing defects. Mainly respondents from the *OFF-THE-SHELF-Software* group consider this as the main reason for delays. Also the testing process can hinder the release of a feature, because “In our case the main reason for delays was the testing process. The testing process took too long and then defects were found, which postponed the process. So it would be nice to have more testers.”, as *PROD-1* states.

4.8 Defects

The main results on Defects include:

(a) Defects are maintained in backlogs and increase the inventory.

(b) Detecting defects in an early stage is less waste.

(c) Removing defects can delay the development process.

The informants handle defects similarly. In most of the organizations the defects are registered and prioritized in the backlog, and depending on the severity level, the defects are fixed right away or taken care of at a later moment. *PROD-1* stated, in his organization there are two defect backlogs for two different teams working on the same product, “this is a bit of a problem.”, because of redundancy. In some of the organizations the backlogs are assessed from time to time to evaluate the status of defects, if they had became invalid, they would be removed them from the backlog. For *PROD-5* there are 3 different levels of defects, “It depends where a defect is found in the cycle. When it’s in development we do not log it and take care of it immediately. In release stage we log the defect and fix it. If the defect was delivered to the customer, there is a strict change process implemented.”

Only *BESP-3* stated they would have a zero-bug policy, which is supported by automated testing and the continuous delivery approach. He also adds, that zero bugs are utopian, but it reflects the attitude towards defects. Furthermore, defects are a *Failure Demand*, which means that a defect can be more than a defect. A failure demand is something that needs attention, because it could have been avoided by setting the right actions in the very beginning, so they evolve over the progress of a project and need to be tracked and eradicated. *BESP-2* shared this opinion and explained it with the high complexity within the system, hence it is more defect prone. So these defects came with poor design in the very beginning.

Informants also argued, that defects interrupt the development process, compare with results in section 4.7. So in some organizations there was a special role introduced to handle defects. This role is reassigned to new developers every sprint, because this is a tedious work for developers, and it’s positive for the individual’s motivation to only be responsible for defects every couple of months. Furthermore, *PROD-3* reported, “The rest of the developers can focus only on stories and new features.”.

4.9 Continuous Improvement

Optimizing the whole is a basic principle in Lean Software Development. It can also be seen as an ongoing effort to eliminate waste. The following results show how organizations have implemented a continuous improvement process and how they utilize metrics to support this process.

4.9.1 Process

The main results on the Continuous Improvement Process include:

(a) Retrospectives are most common used method to improve the process.

(b) Motivation among team-members is an issue, when discussing the same thing all over again.

(c) One improvement at a time is better than parallel efforts.

Most of the development teams perform retrospectives in order to identify problems and improve the process. There are different ways of conducting these retrospectives though. Some of the informants reported to follow this process very strictly, whereas others define it only loosely within the organization. *BESP-1* argues, that “it’s a tedious process and the same old stories are addressed again and again. This lowered the motivation of the employees. This needs to be changed.”. The same happened in *BESP-2’s* organization. It was part of the daily routine, “and nothing really changed, so the team members were not interested anymore. But we are trying to develop it again.”.

Other informants follow continuous improvement more diligently. *BESP-3* reported, “We stop doing things a lot when we feel they are not working.”. In this organization there are different colors for different types of tasks, among them are improvement tasks,

“It’s about improving the development process, solving root cause problems, if we are getting the same issue again and again.”. This also lowers the failure demand, as discussed in 4.8. Furthermore, “Every Friday after lunch, we only work on our company, we do not work for the customers, we work on improving our business”. In *BESP-4’s* department is a special improvement group installed, that maintains an improvement backlog, and everyone is invited to participate in this process. Additionally, retrospectives are facilitated and “We usually pick one improvement task each sprint, that we commit on to improve.”. *PROD-1* argued for the importance of only picking one improvement task at a time as follows, “So based on experience it’s not good to have more. If we try to do 3 or 4 improvements, we usually end up doing nothing.”

In *PROD-5’s* company there are retrospectives conducted on team level, but also on corporate level. There is a dedicated team, that tries to streamline those improvements, so teams can work together cross-functional. Another important part is training of the employees to maintain a high level of technical knowledge.

4.9.2 Metrics

The main results on Continuous Improvement Metrics include:

(a) Metrics are hardly used to support the Continuous Improvement process.

(b) It is difficult to measure desired aspects of the process.

Metrics are not commonly used to make improvements measurable. Most important factor to observe is the lead time of a ticket, as *BESP-3* explained it, “The most valuable metric, for instance in this Kanban thinking, is the lead time. How fast things flow through the pipeline.”. He was again referring to being as fast as possible. This opinion is shared by *PROD-1*, but they are doing it in a less formal way and have no performance indicator defined for that.

PROD-3 pointed out how difficult it is to define the correct metrics, because “We are measuring finished story points in a sprint, but when a story with 15 points is not finished, it is passed to the next sprint, although most of its work happened in the current sprint. So the statistics are skewed.”. *BESP-4* has the same opinion, but “I think there is a lot of potential in doing measurements.”. He also stated the problem of the correct definition of meaningful and useful metrics.

4.10 Difference between BESPOKE and OFF-THE-SHELF Vendors

To begin with, there were no major differences between the investigated groups noticeable that would lead to particular problems within the development process. The following table 4.3 contrasts the subtle differences.

Bespoke Group	**Off-the-shelf Group**
Customer Involvement	
Direct and close contact with customer	Requests are handled Product Owner or help-desk
Defects	
Fixing defects seems to be a bigger issue for Off-the-shelf Vendors	

Table 4.3: Comparison Bespoke and Off-the-shelf

Communication with the customer Communication is very important to prevent certain types of waste, such as extra features or partially done work. This is inhibited by the nature of product development for off-the-shelf vendors. However, these companies do not observe a problem with it as they revise the set of features and try to match it to the wish list, most of them maintain. Hence, the product is streamlined as effectively as possible.

Educate the customer As the bespoke vendors have a the desire for collaboration with the customer and maintain a close relationship with them, they need to get educated in the agile process.

Defects A higher number of Off-the-shelf respondents stated that fixing defects would be an issue for them as compared to Bespoke respondents. This might be attributed to the relatively high amount of different customers using the same product. The frequency and possibility of finding defects is higher the more people use the software. Considering the low number of interviewees, as discussed in section 3.3 it might be also possible that the chosen sample of Off-the-shelf vendors simply producer software of lower quality. The true reason was not identifiable therefore.

Chapter 5

Discussion of the Results

The following chapter presents the discussion to the results in chapter 4. The main purpose of this thesis is to find out how IT professionals perceive waste and to which extent they are aware of waste within the development processes. The most interesting results are discussed in the following. A statement will be given at the end of the following sections regarding the perceived severity of the particular type of waste (sections 5.2 through 5.8). Three different levels of severity are used to make the distinction:

Insignificant The respondents do not perceive waste or inefficiencies within the development process with regards to this type of waste.

Minor Severity The respondents are aware of this type of waste and cope with it one way or another.

Major Severity The respondents are not aware of the influence of this type of waste. This type of waste can be a hidden waste that is not addressed either.

5.1 The informants' definition of the term waste in software development

Waste is basically defined as non-value-adding activities, but the informants have a very wide range in their definitions of waste in software development. Starting with the very short and precise definition of *BESP-4* "Waste is work that does not contribute to the value." coming down to the more detailed and specific definitions of *BESP-2* and *PROD-5*. The respondents obviously reflected about their own organization and gave examples that were reoccurring in the entire individual interviews. This supports the conclusion that the informants perceive waste as obvious problems in their organization, rather than a holistic reflection of the whole.

Considering the number of references to value it can be derived from the results that informants are aware of value in order to deliver value to the customer. BESP-4 had a very clear definition about waste, but did not refer to value for the customer, whereas BESP-3 had a similar view of waste and he is referring to value much more often, which indicates that this organization is much more aware of creating value for the customer than others. This is also supported by the emphasis on delivering as fast as possible and involving the customer as much as possible. This is compliant to the agile principles as described in section 2.1. Furthermore, the rest of the informants do not really comment on value, but they share the opinion of delivering as fast as possible. This leaves the impression that agile processes support the efforts to deliver fast, but do not emphasize the importance of value for the customer.

5.2 Occurrence of Partially Done Work

Partially Done Work does not seem to be an issue among the informants, when it comes to delivering artifacts. The emphasis on delivering as fast as possible indicates, that they try to avoid Partially Done Work automatically, yet unfinished features and non-fulfillment of requirements seem to occur. This is also supported by Middleton et al. (2005) and his claim to pursue a continuous flow. The described problems are probably attributed to miscommunication between developers, product owners and customer. It can be derived that agile processes help diminishing Partially Done Work by focusing on delivering value fast. However the implementation of the communication structure might lead to impediments that need to be addressed.

It seemed as though the informants were not familiar with Partially Done Work, as they needed further explanations on the term in order to reply to the question. This indicates that they do not experience this type of waste regularly. A possible explanation can be found in the utilization of agile processes, that inherently aim on reducing Partially Done Work, in order to deliver value fast by dividing the work load into smaller chunks as proposed by Poppendieck and Poppendieck (2007) to tackle this problem.

Most of the informants maintain defect backlogs. Poppendieck and Poppendieck (2007) consider this as inventory and should be subject to improvements. However, the informants do not perceive this collection of defects as a problem. This clearly contradicts with the Lean Software Development principle “Build Quality In”. It is better to avoid defects in the beginning, than to test quality into the product in late stages. Maintaining a defect backlog leads to context switching, because developers are forced to choose more than one task at a time, depending on the prioritization. Hence, the missing perception or negligence of this type of waste leads to a **Major Severity** within the development process.

5.3 Occurrence of Extra Features

The discussion of the waste type Extra Features was divided into two sections. The results for Customer Involvement are discussed. Furthermore, extra features can be inhibited by interacting with customers closely, as the following discussion shows.

5.3.1 Customer Involvement

Close customer involvement seems to be very vital for the examined organizations. It can resolve problems with extra features and miscommunication resulting in extra work to correct insufficiently implemented features. It seems that the customers need to be educated to fit in the agile process and to allow close involvement. The customers need to be trained, which might be difficult in the beginning. As *BESP-3* states “It is difficult to be agile in a non-agile environment.”. The agile principle of close collaboration with the customer, compare with the agile manifesto (Beck et al., 2001), where customers are closely involved in the development process and give feedback. This helps diminishing the effects of *Extra Features* and can increase the service level. Also a big part of training can be omitted, because the customer is highly involved in the creation process. This way the customer learns how to use the system during the development process. This level of interaction is proposed by Highsmith (2009) and follows the principles described in the agile manifesto (Beck et al., 2001).

5.3.2 Extra Features

Extra feature also occur occasionally in the respondents’ organizations, but it seems not to be a major problem. This issue is mainly attributed to miscommunication or not knowing exactly what the customer asks for. The companies react more or less the same way, they remove features that are of no value for the customer. This can happen very systematically, e.g. *BESP-3*, *PROD-3*. Other informants even think extra features can

be beneficial, when it comes to, e.g. repeat business, as stated by *BESP-1* or gain of knowledge, as stated *PROD-2*. Whereas Poppendieck and Poppendieck (2007) consider this as waste, which does not provide clear value to the business efforts.

Most of the respondents might not even be aware of the importance of this issue. As there is no clear value for extra features, they report, it was not problematic for them. Some see a value for themselves, as they gain knowledge and can generate repeat business by exceeding the expectations. Even though it is considered as waste by Poppendieck and Poppendieck (2007) and hence should be eliminated. However there is no systematic evaluation of the benefits or losses caused by extra features. Hence, the awareness and even tolerance of this type of waste leads to a **Minor Severity**.

5.4 Loss of Knowledge

Poppendieck and Poppendieck (2007) state that it is crucial to create and preserve knowledge, yet they do not present a way that is the most efficient. The distribution of knowledge happens in face-to-face communication, additional to that the informants use methods of visual management, such as KANBAN boards, to visualize the current state within the development cycle, compare with 2.2.8. Furthermore, teams are self-organizing entities and they should decide themselves what degree of documentation they would need. All these methods increase the flow of knowledge within the teams. Experts evolving from the communication and task assignment structure can be critical. In highly complex systems informants rather tend to develop experts, whereas others prefer to distribute the knowledge among the team members equally.

It is necessary to find the right balance. It is crucial that even experts share their knowledge with other team-members occasionally, e.g. in brown-bag sessions. This way the knowledge gets distributed and experts are less distracted in every-day work. Furthermore, the adverse effects of labor turnover are minimized. Hence, due to the support by the applied agile process to preserve knowledge it can be assessed that this type of waste is **Insignificant**.

5.5 Handoffs

As the results in section 4.4 Relearning show, that a close relationship to the customer is beneficial when it comes to preserving knowledge. Also a vital communication process within the team can counteract the loss of knowledge. As with every single handoff tacit knowledge gets lost, reducing handoffs should be aspired. Poppendieck and Poppendieck (2007) claim that only 6% of the tacit knowledge is left after 4 handoffs. The informants reported that there are basically no handoffs within the development process, there is an extra handoff when a separated testing team is involved. Which is not an issue regarding to Poppendieck and Poppendieck (2007) definition of this form of waste. More problems occur prior respectively posterior to the development process, especially when the customer is isolated from the development team. Taking responsibility throughout the whole development pipeline can be seen as a recipe to diminish those effects of a high distance. This also requires a close relationship to the customer. Some of the respondents also apply code-reviews to maintain a higher quality and support the communication process. This is technically not to be considered as a handoff, since knowledge is shared rather than transferred. Hence, the support by the applied agile process to limit the handoffs is beneficial and this type of waste can be assessed as **Insignificant**.

5.6 Task Switching

First of all the tasks are not assigned, but chosen by the team-members. Which is compliant to the definition of a Scrum team as a self-organizing and self-empowered entity 2.1.1. All informants have this in common, which is part of the agile thinking. Also the limitation of the number of tasks to 1 is a collectively shared opinion. A problem can be identified when it comes to the number of tasks, that need to be handled at the same time. Context switching leads to delays, because the developer has to start the thinking process all over again (Poppendieck & Poppendieck, 2007). Those interruptions mostly occur due to higher prioritized activities, such as fixing a defect. Some of the companies have introduced a special role for this, 1 person is responsible for fixing defects in a certain period of time, to counteract this problem.

PROD-5 refers to experts within the team, that are interrupted by less knowledgable team members. This clearly shows a disadvantage of experts evolving within a developer team. *PROD-5* also states that it is necessary to find a balance between knowledgable people that know how to tackle problems within a complex system and distributing knowledge among the whole team. This can be seen as a process, where knowledge should be built up and the distribution of knowledge should go hand in hand. Hence, this type of waste exhibits **Minor Severity**, because the respondents are aware of it and try to deal with it. However, there is still room for improvement left.

5.7 Delays

There are several reasons for delays and those are quite different for the examined groups. The *BESPOKE-Software* group mainly declares impediments from the outer environment as the main reason for delays, whereas the *OFF-THE-SHELF-Software* group sees the interruptions due to defect fixing as the main factor. This is reflected by the work

of Poppendieck and Poppendieck (2007). They state that developer have to make decisions every 15 minutes and are only able to do so, when the most relevant information is present at the time. This difference may be attributed to the lower level of communication with the outer environment. The second group simply is not relying on the customers as much.

Poor design in the beginning leads to a higher degree of complexity within system. This clearly contradicts with the Lean Software Development principle of "Build Quality In" as explained in section 2.2.2. *BESP-2* faces this problem and is well aware of, but he finds it difficult to change the system and simplify it. Whereas *BESP-3* counteracts this problem with a zero-bug policy, where the root cause for a defect is identified and the system is kept clean and simple. In other words, fighting the decease is better than treating the symptoms. This is only possible when the team is able to control the complexity of the software. Some of the respondents have the problem of a high complexity which clearly leads to delays within the development process. Defects, the communication with the customer and the complexity are delaying the process. Hence, the missing perception or negligence of it leads to a **Major Severity** within the development process.

5.8 Defects

Most of the respondents maintain a backlog of defects, which is considered to be Partially Done Work, because those defects increase the inventory. One of the key principles in Lean Software Development though, is to minimize the inventory in order to eliminate waste, compare with section 2.2. An effect of this inventory became very noticeable for these companies, because the management and eradication of these defects took a lot of time and distracted the team members in the course of the development process. Some of the interviewees found a solution by introducing a specific role that temporarily handles all the defects.

This role is reassigned to another team member after some time, because this work tends to be bothersome and tedious. This raises the overall motivation within the team. Yet, the root cause for the defects can not be solved with this measurement. Hence, this type of waste exhibits **Minor Severity**, because the respondents are aware of defects and try to tackle this problem.

5.9 Continuous Improvement

Continuous improvement takes place in an informal manner. The respondents utilize stand-up meetings and retrospectives, however they did not report about a formalized structure for improvements. This indicates the lack of awareness or even denial for problems within the development process. *BESP-2* even stated, that they do not use retrospectives anymore, because the employees lost the motivation for it and it had not changed anything. Yet, he also reported about major problems when it comes to a mutual understanding of the requirements. Whereas *BESP-3* even introduced improvement tasks along with the daily task fulfillment. Hence, there is a different opinion about the importance of continuous improvement among the interviewees.

Making improvements measurable requires defined metrics to compare the state in the beginning and end. The respondents mostly have no metrics defined. This again indicates a certain lack of awareness for improvements. For those who have established metrics utilize them only on a high level, such as task traversing time. They also pointed out that the definition of metrics is very difficult and to appraise numbers might be tricky as well. It is necessary to find the right balance between formality and informality, because too formal processes might impede the openness and agility within a development team, whereas an informal approach might make it hinder the efforts to eliminate waste effectively.

The improvement process should follow the lean principle “Optimize the whole”, as described in 2.2.7, because it is not beneficiary to optimize only single chunks and cut out on the holistic view. This involves the management to act as well as the team-members. The efforts undertaken by the interviewees are commonly restricted to single aspects of the whole. Solely one respondent explained the continuous improvement process as an integral component of the companies business.

5.10 Difference between BESPOKE and OFF-THE-SHELF Vendors

As the work of Poppendieck and Poppendieck (2007) has shown there should be differences between the two interviewed groups in the perception of value for the customer. They argued that BESPOKE software vendors tend to utilize project management structures within their organizations, whereas OFF-THE-SHELF software vendors rather follow a product management approach. Within product development oriented organization value might be market-share or profitability. Whereas, project management organizations value the ability to meet costs, schedule and scope commitment constraints. These two different views of value within the development process were proposed by Poppendieck and Poppendieck (2007) and were not shared by the studied groups. The *BESPOKE*-group in particular is more focussed on delivering value fast in incremental stages rather than meeting time and cost constraints. This enables the development teams to react on changing conditions and adaptability becomes more important than time constraints of fixed requirements. This attitude towards value within the development process is supported by the agile thinking.

5.11 Is relentless elimination of Waste a good approach?

According to Ohno (1988) the highest goal to pursue is “the absolute elimination of waste”. It seems as though that the “absolute elimination” of waste can cause waste in itself. Following this approach relentlessly, might mean that the focus drifts from delivering value to diminishing inefficiencies within the development process. Maybe the Pareto principle, or commonly known as the 80/20 rule, applies here effectively. Addressing the biggest issues (the top 20%) first, can lead to a removal of 80% of the waste. More importantly, it takes a lot of effort to eliminate the remaining 20% of the waste. The 80/20 scale is not fixed. It should solely indicate that little effort can have a big effects and focussing on the wrong issues might lead to tremendous effort.

Chapter 6

Conclusion

Icelandic IT-professionals are well aware of waste in their organizations. When it comes to waste they are confronted with many different problems. It seems though as if they can cope with these problems well and they do not interfere with their ability to deliver fast. Yet, there are some types of waste that impede the development processes. These issues are mainly attributed to communication problems or maintenance.

Most of the respondents maintain defect backlogs. Per definition, this raises the inventory, and hence is considered as partially done work. These defects are prioritized according to their severity. This leads to interruption within the development process. Some companies have introduced a special role to diminish these effects. A member of the development team handles these defects for a limited amount of time before this role is passed on to another person, because of its tedious character. However, the inventory is not reduced by this method, but delays occur much less.

The following statement summarizes the main obstacles within the development process the best.

"It is difficult to be agile in a non-agile environment."

Even though agile processes were applied (mostly Scrum), the customers and other stakeholders from the external environment are not used to collaborate in this agile environment. This is mainly attributed to the disparity in the understanding of collaboration and communication between vendor and customer. Agile teams need a close collaboration and feedback from the customer in order to adapt to changing situations and deliver value fast. It seemed that customers are still more used to the traditional processes, e.g. waterfall model, where a different degree of communication is needed throughout the process. The customer needs to be educated to collaborate in an agile environment. This lack of communication was the main factor for delays and extra features. The negligence of this issue leads might lead to major waste within the development process.

Strong ties and the communication structure within the development teams eliminate several types of waste in the very beginning. For instance, preservation of knowledge was not an issue at all for the investigated companies, although they did not state they would use any other tools than wiki-pages or low-level documentation. The open and respectful face-to-face communication was widely the best method to share knowledge.

The application of Scrum as the development process was beneficial in many cases. For instance, handoffs and the number of tasks are limited by the nature of Scrum. Additionally many companies utilize Kanban to visualize the work in progress. This builds awareness of current tasks and their lead time. These teams are more likely to be able to deliver fast, which is concurrent with the agile principles.

Bibliography

Anderson, D. J. (2012). *Lean software development.* Available from http://msdn.microsoft.com/en-us/library/vstudio/hh533841.aspx

Beck, K., Beedle, M., Bennekum, A. van, Cockburn, A., Cunningham, W., Fowler, M., et al. (2001). *Agile manifesto.* Retrieved March 22, 2013, from http://agilemanifesto.org

Denne, M., & Cleland-Huang, J. (2004). *Software by numbers - low-risk, hight-return development* (Vol. 1). Sun Microsystems, Inc.

Highsmith, J. (2009). *Agile project management: Creating innovative products* (Vol. 2). Pearson Education.

Icelandic chamber of commerce. (2013). Retrieved 15.05.2013, from http://www.vi.is

Kniberg, M., Henrik Skarin. (2010). *Kanban and scrum - making the most of both*. C4Media Inc.

Larusdottir, M. K. (2009). *User involvement in icelandic software industry*. Uppsala, Sweeden: Interact 2009.

Leffingwell, D. (2011). *Agile software requirements: Lean requirements practices for teams* (Vol. 2). Pearson Education.

Middleton, P., Flaxel, A., & Cookson, A. (2005). Lean software management case study: Timberline inc. In H. Baumeister, M. Marchesi, & M. Holcombe (Eds.), *Extreme programming and agile processes in software engineering* (Vol. 3556, p. 1-9). Springer Berlin Heidelberg.

Ohno, T. (1988). *Toyota production system: Beyond lage scale production*. Productivity Press.

Poppendieck, M., & Poppendieck, T. (2007). *Implementing lean software development: From concept to cash (the addison-wesley signature series)* (3rd ed.). Addison-Wesley Professional.

Schwaber, K. (2004). *Agile project manage with scrum*. Microsoft Press.

Shingo, S. (1982). *Study of toyoda production system from an industrial engineering viewpoint*. Productivity Press.

Silverman, D., & Rapley, T. (2011). *Qualitative research* (Vol. 3). SAGE Publications Ltd.

Statistics iceland office. (2013). Retrieved 15.05.2013, from `http://www.statice.is`

Sutherland, J., & Schwaber, K. (1997). *Business object design and implementation : Oopsla '95 workshop proceedings,16 october 1995, austin, texas*. London New York: Springer.

Williams, L. (2012). What agile teams think of agile principles. *Communication of the ACM*, *55*(4).

School of Computer Science
Reykjavík University
Menntavegi 1
101 Reykjavík, Iceland
Tel. +354 599 6200
Fax +354 599 6201
www.reykjavikuniversity.is
ISSN 1670-8539