MASTER'S THESIS | LUND UNIVERSITY 2018

A Model for Value and Cost Trade-offs in Agile Software Requirements Prioritisation

Elin Blomstergren

Department of Computer Science
Faculty of Engineering LTH

ISSN 1650-2884
LU-CS-EX 2018-36

A Model for Value and Cost Trade-offs in Agile Software Requirements Prioritisation

Elin Blomstergren
elinblomstergren@gmail.com

September 27, 2018

Master's thesis work carried out at
the Department of Computer Science, Lund University.

Supervisor: Johan Linåker, johan.linaker@cs.lth.se

Examiner: Björn Regnell, bjorn.regnell@cs.lth.se

Abstract

With the changes that the rise of agile methodologies has brought, the practice of requirements prioritisation has become a central part of the development process. Agile requirements prioritisation focuses heavily on customer needs and implementation time, ignoring many other aspects that will affect the outcome of the software development. This master's thesis aims to identify a wider range of aspects that agile companies should discuss through a case study. The case company develops an open source product and has mostly market driven requirements engineering, resulting in many different opinions to weigh in the prioritisation process.

The case study was conducted through interviews with employees from both the engineering department and management at the case company. Interviews consisted of open questions based on requirements prioritisation literature. Results were analysed and summarised to create the proposed model.

The result from the case study is a prioritisation model based on identified aspect with a discussion on how to apply the model in a company setting. The model describes value-adding aspects of product requirements with the themes *Customers' Needs*, *Product Quality*, and *Timeline*. Cost-adding aspects are described with the themes *Complexity* and *Implementation*. Depending on how the rest of the requirements engineering process is structured, the model can be applied on different levels of the prioritisation process.

Keywords: agile, requirements prioritisation, requirements engineering, dual licensing, prioritisation model

Acknowledgements

I would like to thank my supervisor Johan Linåker for support and feedback throughout my work on this masters thesis. Johan was part of the process from the beginning and helped me find and define my research problem, so without him, this thesis would never have been written.

Furthermore, I would like to thank my supervisor Björn Regnell for feedback on the thesis and insights on how to relate the results to existing research on the topic.

I would also like to thank everyone at the case company who participated as interviewees or just wanted to discuss requirements engineering in the lunch room to give me new ideas on what to research. Finally, I would like to express my gratitude to Johan at the case company who supervised my work and helped connecting me to interviewees throughout the organisation.

Contents

Chapter 1

Introduction

Agile software development is a result of changing business environments during the 1990's. The market demanded shorter time to market and the software requirements changed faster than you could produce working software [31]. As a result, new methodologies that embraced change started replacing the old methodologies that relied too heavily on plans and documentation.

One central concept for agile methodologies is the analysis and prioritisation of changing software requirements [25]. Requirements prioritisation is the process of weighing the values and benefits of a requirement against the costs and risks in order to find out which requirements are the most beneficial for the organisation. When working with limited time and resources, an organisation needs to continuously evaluate and prioritise its product requirements in order to ensure that a high-value product is delivered to the customer. In theory, the prioritisation should be done by having discussions throughout the project where the customer assigns values to the requirements and the developers identify costs and risks. Cao et al. found that in practice, however, many organisations find that continuous communication between customers and developers can be hard to achieve [6]. The responsibility of prioritisation falls on a few individuals that are tasked with understanding the values and costs as seen from other stakeholders' perspective. Another common pitfall found in the same study is that agile practitioners often focus too much on business values and customer perspectives and end up with systems that lack in maintainability, scalability and security.

More and more companies are becoming interested in releasing their software as open source. Focus is shifting from owning important resources in software development towards coordinating these resources [11]. One popular open source business model is dual

licensing, where the company releases both an open source version and a proprietary version of their software. Users of these two versions often have different goals and priorities for the product. Therefore, the company is faced with the problem of gathering opinions from a broad user base, and taking this increasing amount of opinions into account in their prioritisation.

When an organisation lacks concrete guidelines for how requirements should be evaluated and prioritised, every individual involved in the requirements engineering process must form their own understanding of the organisation's values. Requirements prioritisation becomes a subjective process and the results will be depending on who conducted the prioritisation. Organisations can face several issues due to lack of a common understanding of its values. Discussions on requirements prioritisation between different people and teams become more complicated when there is no agreement on what makes a requirement important. New members of the organisation are left out of the requirements evaluation process until they have observed the organisation for long enough to form an understanding of its values. Well-formulated guidelines for how requirements should be prioritised can improve the communication within the organisation. The requirements prioritisation will be more homogeneous and independent on who participated, and the group of potential participants who understand the company values can be broadened.

A case study was conducted at a medium-sized software development company that produces open source software in order to investigate the problems with agile requirements prioritisation in practice. Current requirements engineering practices at the case company were studied and employees were interviewed on their view of how the company should prioritise product requirements. Based on this, a solution for how the case company and similar companies can evaluate product requirements is proposed.

This report aims to investigate following research questions:

RQ1. Which aspects should be taken into account in agile requirements prioritisation for companies that develop open source products?

RQ2. How should these aspects be measured?

RQ3. How can this model be applied in a company setting?

RQ1-3 were answered by studying the case company in four steps. First, the stakeholders of the requirements engineering process were identified and their communication of requirements were mapped out. Second, the aspects that the case company discusses when analysing requirements were identified. Third, these aspects were further analysed and ranked after importance. The fourth and last step was to use the aspects to formulate a model for analysing and prioritising software requirements.

The final prioritisation model is a set of questions to be considered divided into five overall themes: Customers' Needs, Product Quality, Timeline, Complexity and Implementation. Different implementations of the model can be done in the organisation depending on current requirements engineering practices, ranging from a tool to ease communication

to a formal ranking system to ensure that the correct prioritisation decisions are taken. It is concluded that there are many different aspects that are valuable to take into account in requirements prioritisation. If no efforts are made to create routines and guidelines on how it should be done, it risks becoming a very complex process.

The report is divided into 9 chapters. Chapter 2 describes related work that is relevant to understand the context of the prioritisation model. The methodology of the conducted case study is described in Chapter 3. The result of the interviews are divided into two chapters: Description of the current requirements engineering practices at the case company is found in Chapter 4, while the results regarding requirements prioritisation is found in Chapter 5. A proposal for a prioritisation model is formulated in Chapter 6. Chapter 7 discusses the threats to the validity of the case study, while Chapter 8 contains a more general discussion on requirements prioritisation. Chapter 9 summarises the conclusions that can be drawn from the case study.

Chapter 2

Related Work

This chapter will describe related work that is needed to understand the context of this thesis. The main focus will be on Agile Software Development, Open Source Software Development, and Requirements Engineering.

2.1 Agile Software Development

Williams and Cockburn [31] describe the start of the agile movement as a result of changing business environments in the mid-1990's. Traditional software development methodologies relied heavily on long-term plans and documentation, which did not fit well when the pace of change in the business environment increased. New methodologies were adopted to embrace the inevitable changes that occurred during projects in order to avoid having project plans get out of date before the the project was finished. Several different methodologies were developed in parallel and were referred to as *agile*.

The agile movement was inspired by several increasingly popular development methodologies [12]. While the methodologies are beyond the scope of this thesis, two examples are given to clarify the central principles of agile methodologies. *Extreme Programming* focuses on small iterations that implement a small number of features, so called *Stories*, that are picked by the customer [2]. Once the stories have been implemented and put into production, the next iteration starts. *SCRUM* has planned and defined processes for planning and closure of a project but treats the development, so called *sprints*, as a *black box* [28]. Iterative sprints are carried out where each sprint contains development, review and

adjustment of the code over a set period of time. The processes of the sprints are carried out according to tacit knowledge or trial and error.

The underlying values of agile software development was described in the *Manifesto for Agile Software Development* in 2001 [12]. Greer and Hamon [13] describe that while there is no common agreement on exactly what agile software development is, agile methodologies share some identifying properties: The development is done iteratively, developers work in close collaboration with customers and make frequent deliveries, and the methodologies are adapted to handle changes in requirements throughout the development while still being able to quickly deliver working software. Other common properties described by Greer and Hamon is a *continuous design improvement*, *continuous delivery* and *test driven development*.

Agile development teams are often described as *self-organising* [31] [16]. The decision-making authority is spread out through the organisation to a higher degree than in traditional software development methodologies [31]. Executives make the business level decisions while the technical issues are decided on by the developers. The teams are generally cross-functional in order to improve communication within the organisation [4]. Case studies have shown that development teams that have freedom to make their own decisions are well aware of their responsibilities and self-improve throughout their work [16]. When executives step in and take over the technical decision making, the teams become unwilling to self-organise and are less likely to take ownership of their work.

2.2 Open Innovation

The concept of Open Innovation was first described in 2003 by Chesbrough and can be summarised as the idea that *'Not all the smart people work for us. We need to work with smart people inside and outside our company'* [8]. Business strategies that rely on open innovation are referred to as Open Strategies [9]. Chesbrough and Appleyard describe how organisations that apply an open strategy expand their value creation process by searching for innovation outside of the organisation's bounds. However, they also describe the need to take into account that a viable business strategy must be sustained over time, and innovation must be captured in a way that can create value for the business. This must all be achieved without alienating the individuals and organisations outside of the own organisations that participated in creating the value.

2.2.1 Open Source Software

A growing number of companies choose to work with open source software development, where independent developers contribute to a company's code base. It has been described as an extreme form of Open Innovation [14]. With the growth of open source software, focus has changed from owning important resources in software development towards co-

ordinating these resources [11].

There are many benefits for using an open source business model. The company receives bug fixes and new code to improve their product from independent developers [10]. Open source can also be seen as a distribution channel by creating interest and trust of the product in a wide community [10]. There are many potential motivations behind participation for the contributing developers. Studies have shown that many view it as a creative exercise and a chance to improve their programming skills [22]. Others are motivated by needing to improve or add to the code for their own user needs [10]. For example, many contributors to open source projects are hired by external firms to contribute open source projects that the firm make use of.

A company cannot simply open their source code and expect a community to form around it. There are many challenges that a company must face in order to create and gain from a community. Dahlander and Magnusson identified three themes that organisations must consider to capture innovation when working with open source software communities: *Accessing*, *Aligning*, and *Assimilating* [11]. Accessing refers to the organisation's ability to access the development in the communities, either by giving incentives to work on the organisation's own product or by identifying external open source resources to use in the product. Aligning refers to the ability to align the goals of the organisation with the goals of the community. Important steps in this is to clarify ownership by licensing the product and to influence the direction of development to ensure that the organisation can reach their goals. Assimilating refers to the ability to integrate results from an Open Source Community to the organisation's code base and to feed back results to the community.

2.2.2 Dual Licensing

Dual licensing is a popular open source business model where a product is licensed under both an open source and a proprietary license depending on how the customer plans to re-distribute the software [11] [9] [10]. Using dual licensing for a product helps the company gain the benefits of open source software while still making money from their product [11]. However, opening the source code of the product comes with the risk of stimulating new competition [10].

A common strategy is to release an open source version that is free to use, but is licensed under a restrictive license [10]. Restrictive licenses force users to release their own software that uses the open source software under an open source license. A proprietary version is released which allows the users to release their own software as proprietary, thus keeping the ability to avoid opening their software to the general public. Obtaining the proprietary license requires the user to pay a fee. It is common that the version that is licensed under a proprietary license has more features, offers a more stable service, and includes support from the company selling the license [9].

2.3 Requirements Engineering

The process of identifying, modelling, communicating and documenting software requirements for a system is referred to as *requirements engineering* [25]. Depending on the context of the software development, there are different approaches to requirements engineering. This section will focus on how requirements engineering is done in market driven development and in agile development.

2.3.1 Market Driven Requirements Engineering

Requirements engineering for software that is sold off-the-shelf to a mass market is referred to as *market driven requirements engineering* [26]. Since the customer segments are spread across the world, there is no single customer that can interact with the company and participate in eliciting requirements. This forces companies to use different elicitation techniques to gather requirements both from different customer segments and internally within the own organisation. Companies that focus too much on their own innovation processes and invent their own requirements face issues when searching for customer bases [20]. At that point, the company most likely has to adapt the product to fit actual customer needs. Market Driven Requirements Engineering also puts the producing company in a bigger financial risk than when developing for specific customers: Since there is no contract with a customer that will pay for the finished product, the organisation that develops the product takes all the financial risk during development [26]. There is often a heavy focus on a short time to market for the software and later regular releases of new versions when the product is established on the market.

Karlsson et al. identified a set of challenges with market driven requirements engineering through interviews with practitioners [20]. Among these challenges were *Communication gap between marketing staff and developers*, *Managing the constant flow of new requirements*, *Requirements volatility*, and *Requirements are invented rather than discovered*. These challenges were shared between several companies of different sizes and domains with the common factor being market driven development.

2.3.2 Agile Requirements Engineering

Agile requirements engineering practices are often less formal than traditional requirements engineering practices, and do not always include documentation of requirements [4]. They rely heavily on customer involvement in defining and prioritising requirements in order to be successfully carried through [25]. Instead of being conducted at the start of product development like in a traditional waterfall model, agile requirements engineering is present in each iteration during development. All of these practices conform with the values expressed in the agile manifesto [12].

Since there are many different agile development methodologies, there is not a single, common way of conducting agile software development. Through a literature study, Inayat et al. could summarise a set of practices that were commonly described in agile requirements engineering literature [17]. The most frequently described practices were the following: *Requirements Prioritisation* that is done in each iteration of the development in order to identify the most important software requirements. *Testing before coding* to get feedback from the pre-written tests failing during development. *Face-to-face communication* between customers and development team members in order to understand the software requirements without the need of heavy documentation. *Customer involvement* in order to identify, clarify, and prioritise requirements continuously. *Iterative requirements* that emerge during the development process as stakeholders interact with each other and the evolving system. *Retrospectives* where developers and customers can meet after each iteration to review their progress and to plan further development.

Case studies have shown that agile requirements engineering practices can help organisation overcome common challenges that occur during traditional requirements engineering. Agile software development methodologies focus on improving communication, which makes the requirements processes more efficient [4]. A case study by Bjarnason, Wnuk and Regnell points to several improvement including bridging communication gaps, preventing overscoping, and avoiding resource waste from requirements being dropped during the development process [4]. They also point out that introducing agile requirements engineering practices requires changes of the mind-set of all team members involved. There must be a shift from assigning requirements engineering to specific people to instead involving the whole team. Agile requirements engineering practices also involve continuous validation through the prioritisation that is done each iteration [17].

Studies point to several challenges that organisations face when adopting agile requirements engineering practices. The lack of documentation causes problems when there are sudden vital changes in requirements or when the people involved are unavailable [17]. One common reason for unavailability is that having customer representatives always on site and ready to discuss with developers is seldom viable. Budget and schedule estimations are negatively affected by the volatility of agile software requirements and dynamic planning [6]. Agile software requirements can cause developers to create an inappropriate architecture that later proves unable to support vital aspects that were not brought up by the customer in early iterations [6]. For example, this can occur with security requirements or non-functional requirements. System level issues can be left uncovered until late in the development process due to the system design being completed late in the life cycle [4]. The need for immediate value creation for the customer can lead to activities such as refactoring being considered unimportant despite providing much value in the long term [5].

2.3.3 Requirements Prioritisation

Software development is often conducted under both budget and time constraints, and it is often not possible to implement all planned features by a scheduled delivery date [30].

In order to solve this and to deliver the most essential functionalities first and to plan any future releases, product requirements should be prioritised. According to Wiegers, the relative importance of each functionality must be established so that releases can be planned to provide the greatest product value for at the lowest cost. Achimugu et al. highlight that requirements prioritisation is a complex multi-criteria decision making process where there can be several relevant stakeholders whose perception of importance must be taken into account [1].

Finding the relative importance of software requirements requires methods for measuring values and costs. Berander and Andrews present three different types of measurement scales: Ordinal, Ratio, and Absolute scale [3]. Ordinal scales mean that requirements are ordered to show which requirements are more important than others but there is no measurement for how much more important they are. Ratio scales show how much more important one requirement is than another, commonly on a scale from 0-100 percent. Absolute scales assign absolute numbers to requirements, like hours of work, to order requirements according to their importance. Ordinal scales are the least powerful and give the least information on the relative priorities of the requirements, while absolute scales are the most powerful [3].

The requirements engineering community has created a number of different requirements prioritisation techniques adapted to fit different organisational needs [15]. A literature study by Achimugu et al. included 73 studies and found a total of 49 described prioritisation techniques [1]. The five most cited and utilised techniques were *Analytical hierarchy processs*, *Quality functional deployment*, *Planning game*, *Binary search tree*, and *$100 allocation*. These techniques are described in more detail in the next five sections.

Analytical hierarchy process, or *AHP*, is conducted by comparing all possible pairs of requirements [3]. It is determined which requirement is more important, and to what extent. Berander and Andrews suggests using a scale of one to nine with one signifying the requirements being equally important and nine signifying one requirement being much more important than the other. AHP is an example of a ratio scale. Prioritising n requirements requires $n(n\text{-}1)/2$ comparisons, making AHP a time consuming technique that does not scale well with big amounts of requirements [1] [19]. The technique is however very trustworthy since the amount of redundancy in the pairwise comparisons works as a consistent check for judgement errors [19].

Quality functional deployment, or *QFD*, is a concept for translating the customer's needs into appropriate requirements priorities [7]. The goal of QFD is to ensure that a high enough customer satisfaction is met. QFD is conducted by using a goal-domain matrix where the customer's goals are mapped to which domain the issue affects [23]. It also includes information on the business value of meeting a goal, how this value is spread between domains, the cost of supporting a work area, and the business value of the functions in a work area. Using a QFD matrix to prioritise requirements provides a robust method for understanding how requirements are related to the customer's needs and values [30].

Planning game is a central part of *Extreme Programming* and is used to help decide what should be developed [21]. Customers classify requirements as belonging to one of three

categories: Those without which the system would not function, those that are less essential but provide significant business value, and those that would be nice to have. Developers estimate implementation time for the requirements and classify the risk of the implementation using three categories: Those that they can estimate precisely, those that they can estimate reasonably well, and those that they cannot estimate. These classifications are then used to let customers pick sets of requirements, calculate potential release dates of the sets, and decide which requirements are to be included in the next release using the estimates. Since the technique does not measure how much more important a requirement is than another, Planning game is an example of a ordinal scale.

A *Binary search tree* in requirements prioritisation is a special application of a binary search tree where each node represents a software requirement [19]. In a binary search tree, all nodes in the left subtree of node X are of lower priority than X itself, while all nodes in the right subtree of node X are of higher priority than X. This allows for listing all requirements in the binary search tree in sorted order by traversing it in order. The search tree is constructed by starting with one single requirement as the top node in the search tree. The subsequent requirement is placed in either the left or the right subtree depending on its importance. Inserting the remaining requirements in done by comparing it to the base node, then to the node in either the left or right subtree, and continuing this until the correct position in the search tree is found. A Binary search tree is an example of an ordinal scale.

$100 allocation is performed by asking stakeholders to distribute 100 units of for example money or hours between a set of software requirements [3]. The technique is and example of a ratio scale and is sometimes called *Cumulative voting* or *100-dollar test*. For greater amounts of software requirements where $100 has not been enough to distribute, researchers have achieved good results by scaling the amount to for example $100,000 [26]. One problem with using $100 allocation with several stakeholders is that can be hard to detect when a stakeholder uses a "shrewd tactic" to influence the final prioritisation [26]. For example, a stakeholder could choose to allocate all resources to a small amount of requirements that will benefit themselves and not allocate any resources to important requirements that they know other stakeholders will prioritise. This risks creating a set of priorities that do not correspond to the real life importance of the requirements.

Despite having a large number of prioritisation techniques that are thoroughly described in literature, requirements prioritisation in a real life setting is seldom a straightforward process. Karlsson et al. list several issues that affect the ability of project management to establish the importance of product requirements: *Availability of resources*, *project milestones*, *conflicting stakeholder opinions*, *market opportunities*, *technological risks*, and *product strategies* [21]. Another study conducted by Lehtola et al. identified five challenges that affected how requirements prioritisation was carried out at two case companies. The five challenges were summarised as *Requirements prioritisation is an ambiguous concept*, *Prioritisation practices are informal and dependent on individuals*, *Requirements are prioritised in many phases*, *Developers do not know enough about customer preferences*, and *The priority of a requirement is based on many factors* [24]. The same study also discussed the issues using scales to determine priorities, as scales such as High-Medium-Low have very different meaning to different people. Using scales required much effort

and discussion in order to unify the team's understanding of what the scales should signify. Regnell et al. pointed out that using absolute scales to determine the importance of requirements is often very hard for humans [26]. They suggested that pairwise comparisons lead to more reliable results, as relative judgements are easier to comprehend than absolute scales. Pairwise comparisons are however much more time consuming to use in requirements prioritisation [1].

Chapter 3

Methodology

To investigate RQ1-3, a case study was conducted with the purpose of finding a potential model for requirements prioritisation. This chapter will introduce the case company, describe how the case study was conducted, and present all interviewees that have participated in the case study.

3.1 Case Company

The case company is a software development start-up company that was founded in 2007 in Sweden. The main product delivered by the company is a database which can be used in a wide range of domains. Today, the company has several offices in both Europe and North America and has around 200 employees globally. Not all employees are working from offices but instead work remotely from a number of European countries and US states, as well as New Zealand and Australia.

Due to a diverse collection of supported use cases, the customers of the company operate in a wide range of industries. The software is distributed both as a Enterprise version licensed under a commercial license and an Open Source version that is licensed under GPLv3 and AGPLv3. An active Open Source community has formed around the company and contributes mainly through bug reports and building add-ons for the product.

The company aims to follow agile principles. Some key concepts that permeate the organisation are flexibility, ability to react quickly to change, and preferring communication over documentation. Key quality characteristics for the product is correctness, robustness,

and reliability. The company also values its ability to take customer requests into account in their daily work and create customised solutions when needed.

A potential prioritisation model for product requirements must be useful for requirements of different levels and types. Due to the agile nature of the work processes at the case company, the model should support easy re-prioritisation as new requirements are discovered.

3.2 Model design

The intended purpose of this model is to identify a wider definition of *Values* and *Costs* in requirements prioritisation. It should be easy to use and to apply within different parts of the organisation, as requirements prioritisation is often carried out in many different phases [24]. Some design decisions that would permeate the formulation of the model were therefore made.

Requirements prioritisation is an ambiguous concept and have many different meanings [24]. In this thesis, it will mean the process of analysing and prioritising requirements in order to plan which requirements are to be included in which release. The model must be easy to use in order to fit the needs of the case company, but should give as much information as possible on the priorities of a requirement, and should scale well with increasing number of requirements. This meant that pairwise comparisons should not be used, as it is both time consuming and does not scale well with greater amounts of requirements [26] [1]. The priorities should not be determined on an ordinal scale as this gives no information on how much more important a requirement is than another [3]. Since requirements practices are very dependent on which individuals are involved [24], it is important that the model is clear on which questions are important to discuss in the prioritisation process. This can hopefully help with unifying the organisation's understanding of what makes a requirement important.

3.3 Case Study

The case study was divided into four distinct phases:

- Stakeholder Analysis
- Aspect Identification
- Aspect Ranking
- Formulating Prioritisation Model

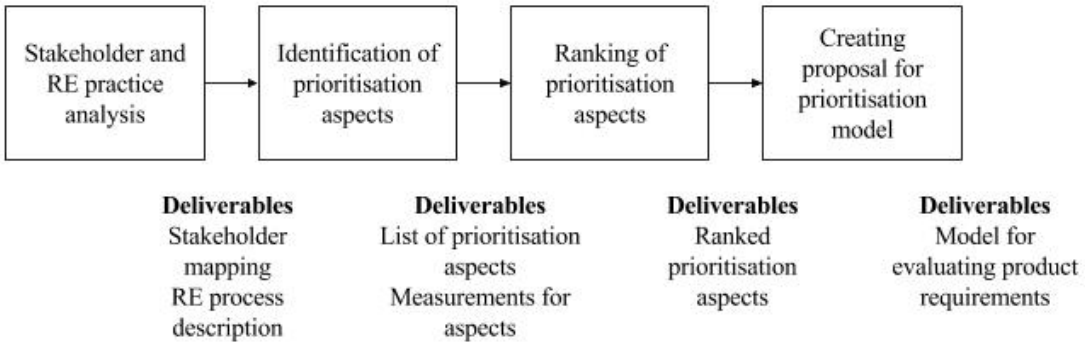

Figure 3.1: Each phase of the case study and the deliverable connected to each phase

Each phase was validated together with an executive in order to continuously evaluate if the work described the case company correctly. The final prioritisation model was evaluated by testing it on real product requirements from the case company.

The following sections will describe each phase in greater detail.

3.3.1 Stakeholder Analysis

The first phase focused on mapping out the organisation to gain an understanding of the requirements engineering organisation. All stakeholders of the requirements engineering process were identified and their communication of product requirements was mapped out. This mapping had three purposes; First, it showed which kinds of requirements were present in the process and which information was available from different stakeholders. Second, it mapped out the current requirements engineering processes of the case company. Third, it showed where requirements prioritisation should be conducted and which actors should be involved in the process.

The methodology presented by Sharp, Finkelstein and Galal [29] was used to ensure a thorough analysis of all requirements engineering stakeholders. The methodology proposes that in order to identify all relevant stakeholders of the requirements engineering process, the analysis should start with *baseline stakeholders* that are either developers or decision makers. The rest of the stakeholders are identified based on their relationship with baseline stakeholders: *Supplier stakeholders* are stakeholders that supply requirements to the baseline stakeholders, *Customer stakeholders* are stakeholders to which the baseline stakeholders supply requirements, and *Satellite stakeholders* are stakeholders that neither supply to nor are supplied requirements by the stakeholders but still interact with the baseline stakeholder regarding requirements. For this case study, the analysis was conducted first with Product Engineering as baseline stakeholder and then with Product Management as baseline stakeholder. This methodology was used in interviews with Product Management in order to identify all relevant stakeholders.

3.3.2 Aspect Identification

The second phase consisted of interviews that aimed to identify the aspects are discussed within the case company when prioritising requirements. These interviews will be referred to as *Aspect Interviews*. During the interviews, the interviewees were asked to describe which aspects they perceived were often discussed during requirement prioritisation. The interviewees were also given a list of seven generalised prioritisation aspects from Berander and Andrews [3]. The purpose of using general aspects was to inspire the interviewees to think of prioritisation aspects in a broad perspective and to give them a context instead of asking them to brainstorm aspects on their own. The words on this list were:

Importance	What aspects could make a requirement important?
Penalty	What are potential penalties if a requirement does not receive high enough priority?
Cost	What can affect the cost of implementing a requirement?
Time	Which time aspects need to be considered when implementing a requirement?
Risk	What risks can you face related to implementing, or not implementing, a requirement?
Volatility	What aspects could cause changes of the requirement?
Competition	How do competitors affect the priority of a requirement?

After all aspect interviews were done, similar aspects were grouped together to lower the number of aspects for further analysis. The original aspects were later used to formulate questions to use when measuring the aspect groups in the prioritisation model.

3.3.3 Aspect Ranking

Interviews were conducted during the third phase to validate if the previously identified aspects were representative for the case company's requirements prioritisation and to rank their relative importance. These interviews will be referred to as *Ranking Interviews*. The interviewees were asked to read through a list of the aspect groups from the Aspect Identification phase and then order them as either "Very meaningful aspect", "Meaningful aspect", "Not very meaningful aspect" or "Unimportant for my team" based on how meaningful they perceived this aspect to be in the requirements prioritisation related to their team. If interviewees stated that there was a difference between their opinion on the most beneficial ranking of aspects and how the ranking is currently done in practice, they were asked to pick the ranking they thought to be most beneficial.

All aspect groups were ranked based how many times an aspect was mentioned in the aspect interviews, its ranking during the ranking interviews, and comments from the interviews. The aspects with the lowest ranking were concluded to not be meaningful during

prioritisation, and were not included in the next phase.

3.3.4 Formulating Prioritisation Model

The last phase consisted of using the aspects, measurements and rankings to formulate a prioritisation model. Each aspect group was classified as either adding a value to or bringing a cost to the product. Value adding aspects could for example ensure that existing customer continue to use the product, attract new customers, or keep the product up to date with new technology. Aspects that brought cost to the product could for example be time consuming, or add risk or uncertainty during development and deployment of the product.

Validation of the prioritisation model was done through separate discussions with two interviewees that have roles that are central in the current requirements prioritisation process. Both interviewees were asked to read through the model beforehand and reflect on if something in the model did not represent company values regarding the software requirements, or whether anything was missing. A validation meeting was held where the interviewees presented their opinions and could ask questions regarding the model. During the meeting, the interviewees were asked to describe how they thought the model should be used to benefit the prioritisation process. All results from the validation meeting were written down and used to improve the model.

3.4 Interviews

11 individuals were interviewed during the interview process and a total of 16 interviews were conducted. The interviews were informal discussions that were based around themes as described in previous sections. During the interviews, the discussions were documented by the researcher through notes. The researcher went through the notes immediately after the interviews to ensure that no information was missing. If any information needed to be clarified, the interviewees were contacted and asked for a new description of their opinion.

Interviewees were chosen to ensure that the opinions of executives, Product Management and Product Engineering were represented. While all interviewees were chosen during the first interview cycle, most interviewees were only available to participate in one of the interview cycles due to other commitments. A short summary of all interviewees that participated in the case study can be found in Table 3.1.

Table 3.1: All interviewees that participated in the study. Listed for each interviewee is their role, how long they have worked for the company, and which interview cycle they participated in.

Name	Role	Aspect interviews	Ranking interviews
I1	Executive that co-founded the company.	Yes	Yes
I2	Executive that co-founded the company.	No	Yes
I3	Executive that has been with the company for 6 years.	No	Yes
I4	Product Manager that has been with the company for 8 years.	Yes	Yes
I5	Director of Engineering that has been with the company for 7 years.	Yes	No
I6	Developer and team lead that has been with the company for 3 years.	Yes	Yes
I7	Developer and team lead that has been with the company for 3 years.	Yes	No
I8	Developer and team lead that has been with the company for 3 years.	Yes	Yes
I9	Developer that has been with the company for 3 years.	Yes	No
I10	Quality Assurance engineer that has been with the company for 2 years.	Yes	No
I11	Quality Assurance engineer that has been with the company for 5 years.	No	Yes

Chapter 4

Requirements Engineering at Case Company

The requirements engineering processes at the case company depend heavily on the agile principles of "Individuals and interactions over processes and tools" and “Working software over comprehensive documentation” [1]. Communication of requirements between different units of the company relies on communication between individuals, and each unit is responsible for their own internal documentation. This chapter will describe the requirements engineering in greater detail.

4.1 Stakeholders

The requirements engineering process at the case company involves a number of units that are either internal or external to the company. These stakeholders are either sources of product requirements, or they relay other units’ product requirements to units with ownership over the product vision and roadmap.

Internal stakeholders are stakeholders that are employed by the case company. They are either developers and designers of the product, or they are internal representatives of the users of the product. In alphabetical order, the internal stakeholders are:

[1] Source: http://agilemanifesto.org/

Customer Success	Keep continuous contact with Paying Customers and Field Engineering to ensure that customers get the support needed to get the most out of using the case company's product.
Developer Relations	Responsible for continuous contact with the Open Source Community. They support Community Users in their use of the product and communicate feedback from the users back to the case company.
Executives	Consists of senior management at the company with a responsibility for the long term plans and vision of the company.
Field Engineering	Engineers that work closely with enterprise customers to create customised solutions and provide support.
Product Engineering	The developers of the product. The developers are divided into teams based on which functionality of the product they work on.
Product Management	Plan the development of the product and are responsible for eliciting product requirements and communicating these to the developers.

External stakeholders are stakeholders that are not employed by the company. They are either users of the product, or they have market knowledge that the case company lacks. In alphabetical order, the external stakeholders are:

Community Users	The Open Source Community that consists of developers who use the product and contribute with an "ecosystem" around the product.
Paying Customers	All customers that use the Enterprise version of the product.
Experts	External experts that the case company consults when starting new projects. This unit is not an actual, consistent group of people. It is independent people with expertise within an area that the case company is unfamiliar with. Therefore, experts will vary from project to project. Product Management describes the use of experts in the requirements elicitation as gaining an understanding of the barriers to enter a market and "avoiding reinventing the wheel".

All stakeholders of the requirements engineering process were mapped to understand how requirements are handled. A mapping of the organisation and how the requirements move through the organisation is shown in figure 4.1.

The graph shows that while there are many sources of requirements, they will all reach

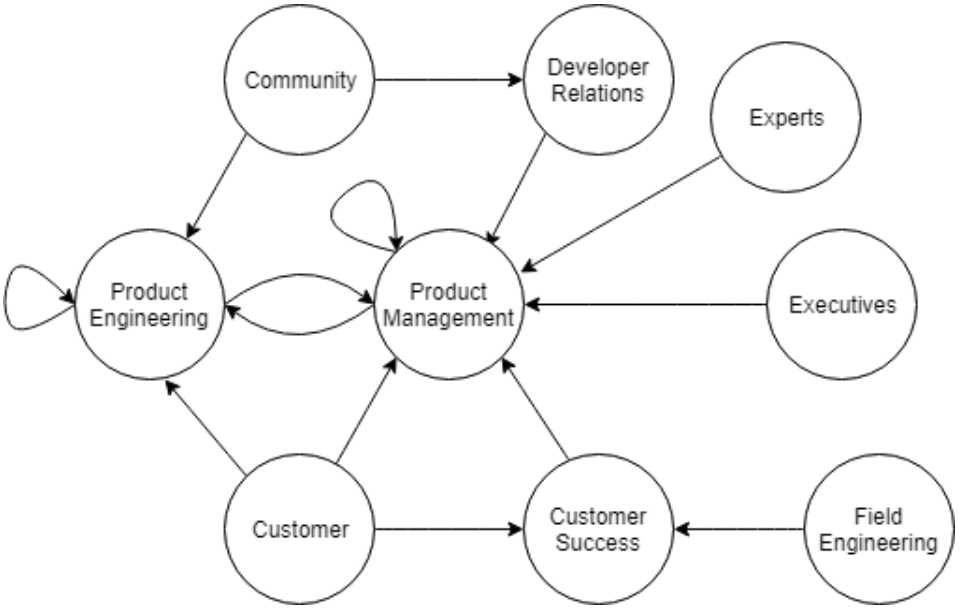

Figure 4.1: A mapping of the requirements engineering stakeholders and how requirements are communicated through the organisation.

either Product Management or Product Engineering in the end. This implies that requirements prioritisation should be done by these two groups.

4.1.1 Licensing and Customer groups

The case company uses dual licensing for their product and this has created two distinct types of customers: Paying Customers and Community Users. These customer groups have different relationships with the company and their input to the requirements engineering process are handled separately.

For Paying Customers, the case company has a proprietary version of the product that is licensed under the company's own commercial license. This version is referred to as *Enterprise Edition*. Obtaining a license requires a customer to communicate with the Sales and Marketing departments within the company, and entails regular contact with Field Engineering and Product Management during the duration of the license. This gives a paying customer a well established relationship with the case company and an infrastructure for communicating requests for product improvements.

The Open Source version is licensed under the restrictive licenses GPLv3 and AGPLv3. This version is referred to as *Community Edition*. Community Users seldom contribute code to the core product and have instead created a self sustaining ecosystem of add-ons around the product. Most communication with the case company goes through Developer Relations who decide if they handle a request themselves or if it should be forwarded to Product Management. Unlike the Paying Customers, Community Users have no direct

contact with the departments that own the product vision and roadmap. This has resulted in Community Users having less input in the high level requirements than the Paying Customers.

Dual licensing is viewed by the case company as having a fair cost for the product in that customers can pay with either money or participation. Participation can mean code contribution and bug reports, but it can also be to spread awareness of the product. The Open Source Community is viewed as a driver of adoption of the product, with the hope that users of the Open Source version will become Paying Customers in the future. This is an incentive for the case company to maintain a good relationship with Open Source Community users that currently do not contribute with neither code nor revenue.

4.2 Requirements Types

Product Management at the case company describe product requirements as belonging to one out of four distinct categories. These different categories differ on where the requirements originate and their scope. The categories can be described as *Visionary requirements*, *Customer requirements*, *Technical requirements*, and *Bug fixes*. A visual representation of how different requirements types are communicated through the company can be found in figure 4.2.

Visionary requirements	Requirements that relate to the roadmap and long term vision of the product. These requirements can come from both executives of the case company and the Product Management.
Customer requirements	Proposals from customers about features and changes that they think would make the product more useful to them. These requirements can vary in scope depending on which customer has requested it.
Technical requirements	Requirements regarding technical opportunities that can improve the product. This type of requirements are almost exclusively discovered by developers with a deep technical understanding of the product and the code base.

Bug fixes	Fixes for bugs reported by users of the system. Both Community Users, Paying Customers, and Product Engineering report bugs. Bug fixes are generally small tasks that are easy to fix, and as a result do not require the attention of Product Management. If bigger issues are discovered during the bug fix, it should be classed as a Technical requirement since it should be part of Product Management's planning.

Interviews with Product Management clarified which types of requirements came from which stakeholder. This process is described as a graph in Figure 4.2. Visionary requirements are mainly elicited from either Executive Management or Product Management and is communicated to Product Management for analysis and prioritisation. Customer specific requirements stem from Paying Customers and are communicated to Product Management either directly or through Customer Success, depending on the customer's relationship to the case company. Technical requirements stem from stakeholders that work with the source code. Field Engineering communicate technical requirements to Product Management through Customer Success. Community Users communicate technical requirements through Developer Relations, who communicate both requirements from the community and their own technical requirements to Product Management. High level technical requirements are analysed and prioritised by Product Management, while low level technical requirements are communicated to Product Engineering. Product Engineering elicit their own technical requirement and communicate any high level technical requirements back to Product Management. Bug reports are mainly submitted by Community Users, Paying Customers and Product Engineering but are communicated directly to Product Engineering without the involvement of Product Management.

Figure 4.2 shows that requirements that reach Product Management is mainly Visionary, Customer Specific, and high level Technical requirements. The requirements that reach Product Engineering are Bug fixes and Technical requirements.

4.3 Requirements Documentation

Requirements engineering at the case company relies more on communication between individuals than the use of tools and documentation. There are no guidelines on how the documentation should be done so Product Management and Product Engineering are free to decide themselves how it should be done. Generally, the documentation is meant for personal use and to keep track of ones own work rather than communicating to other departments and teams what is currently being worked on.

Product Management document requirements on a roadmap description level. These requirements are meant to give an overview over one or more release cycles and do not regularly give much detail on the design and implementation of the requirement. Prod-

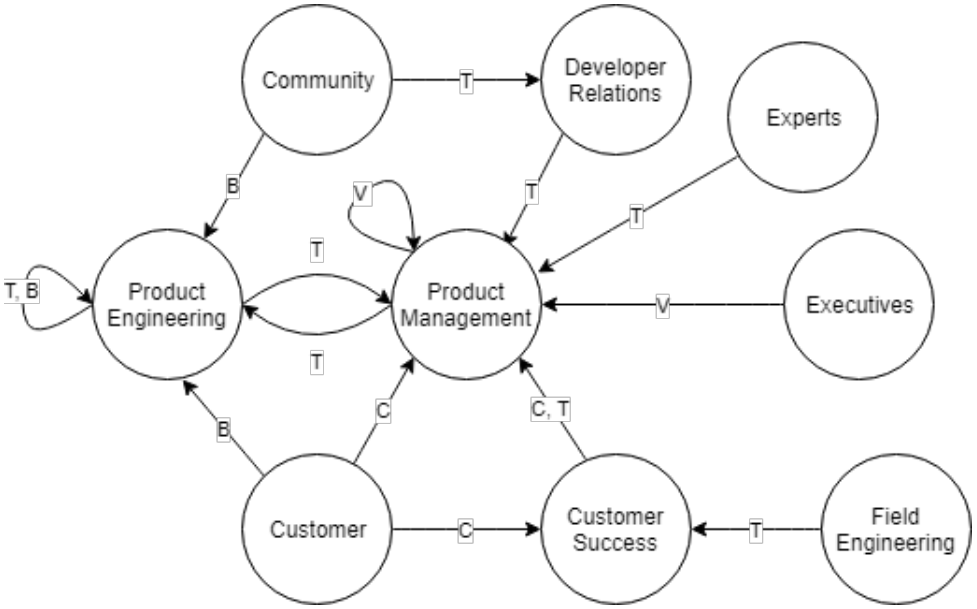

Figure 4.2: A mapping of which types of requirements that are communicated between stakeholders. V stands for Visionary, C for Customer Specific, T for Technical, and B for Bug Fixes.

uct Engineering documents requirements as smaller tasks and use them to keep track of their daily work. These requirements consist of technical descriptions that can be hard to interpret for someone without an understanding of the source code.

No dedicated product management tool is used to keep track of all requirements. Instead, whichever tool is deemed to fit best to describe the requirement is used. Some common examples are text documents, task cards on a digital board, or presentation slides.

4.4 Requirements Prioritisation

As seen in figure 4.1, product requirements converge at either Product Management or Product Engineering, which would make these two units ideal for collecting, evaluating and prioritising requirements. This is also the routine today.

Product Management work with Visionary, Customer Specific, and some Technical Requirements, as seen in figure 4.2. The requirements are high-level and describe features or user stories rather than technical details. Prioritisation on this level is done for long term perspectives such as one or several release cycles. While this prioritisation is not subject to daily changes, it will be re-evaluated during the release cycle and new requirements can be added. This requires any model for prioritisation to support easy re prioritisation. The prioritised high level requirements are relayed to Product Engineering, where they are broken down into smaller tasks by the teams of developers.

Product Engineering work with low level requirements such as detailed Technical require-

ments and Bug fixes, as seen in figure 4.2. The day-to-day work for developers consists of continuously prioritising which issues need to be handled and often depend on either urgency of the need or technical dependencies in the code. Product Engineering is also tasked with breaking down Product Management's prioritised requirements into smaller tasks. For these requirements, prioritisation of features is already done by Product Management and the prioritisation of smaller tasks is mainly based on technical dependencies.

The case company does not have any concrete guidelines for how requirements prioritisation should be made. Instead, it relies on every actor being well informed about benefits and costs to make a decision. Generally, benefits mean business values, and costs mean man-hours to implement a feature and to which degree the work can be parallelised. This method of prioritising has worked well for the case company this far. However, there are some problems that are also becoming more noticeable as the company grows. The prioritisation is a very subjective process and the results differ depending on which individuals were involved in the process. When hiring new employees, it is complicated to communicate the company values and what aspects are seen as benefits. It is also hard for other teams such as Product Engineering to give input into the prioritisation process and a lot of technical insight is lost.

4.5 Use of Prioritisation Model

The purpose of the case study is to create a prioritisation model that is easy to use but still produces at better result than current prioritisation practices. To ensure this, the model must fit with the rest of the requirements engineering process. The following considerations should therefore be taken into account:

The requirements that are to be prioritised are high-level requirements that will span over a release cycle. Requirements are not yet described in detail and the model must not rely exact feature designs.

Each requirement cannot be too time consuming to evaluate. Release cycles cover a large amount of requirements, and it is not viable for the planning and prioritisation phase to take too much time. A prioritisation model should be a basis for discussion where the depth of analysis can be adapted to how much time is available.

Since many different types of requirements are handled similarly at the case company, the prioritisation model should support all different types. Different types of requirements have a value for the company for different reasons. This means that the model must use a wide definition of "Value" to ensure that no requirements type gets an advantage over the others.

In the prioritisation process, Product Management take on the role of representing the customers and end users of the product. It is not possible for the prioritisation model to rely on extensive feedback from customers. The available information will be business values and product vision from Product Management, and cost and risk estimates from

Table 4.1: A summary of the challenges that the proposed prioritisation model must handle.

ID	Description
C1. No Guidelines	Since there are no concrete guidelines on how prioritisation should be made, it will differ depending on who is involved in the process.
C2. Hard to Communicate	Values for prioritisation are hard to communicate to new employees or employees that have previously not been involved in the prioritisation process.
C3. Lack of Documentation	The case company does not rely on documentation. Requirements that are to be prioritised are high-level and not thoroughly documented and designed.
C4. No Customer Present	No customer and end-user can actively participate in the requirements engineering process and the model cannot rely on extensive customer feedback.
C5. Requirements Types	There are many different kinds of requirements types that need to be handled in the requirements prioritisation process.
C6. Many Requirements	Each time requirements are prioritised, there are a big amount of requirements to analyse. The prioritisation process cannot be too time consuming for each requirement.

Product Engineering.

This chapter has identified a set of challenges that the proposed prioritisation model should handle. These challenges are summarised in Table 4.1.

Chapter 5

Results

This chapter will present the results from the two interview cycles. In 5.1, all identified aspects are described. In 5.2, measurements for each aspect are proposed. In 5.3, the relative importance of each aspect based on both mentions in the aspect interviews and the ranking in the ranking interviews are presented. A short summary on all identified aspects and their rankings is presented in 5.4.

5.1 Aspects

The interviewees identified 26 different aspects as important when discussing and prioritising requirements. These are referred to as sub aspects (SA), to separate them from our grouping of aspects that are later used in the prioritisation model. Figure 5.1 shows all sub aspects and how they are related to the generalised prioritisation aspects *Importance*, *Penalty*, *Cost*, *Time*, *Risk*, *Volatility*, and *Competition* that were described in Chapter 3.2.2.

In order to make the aspects more comprehensible and to decrease the number of aspects, the sub aspects were grouped into 15 aspect groups (A). All sub aspects and aspect groups are listed below in alphabetical order.

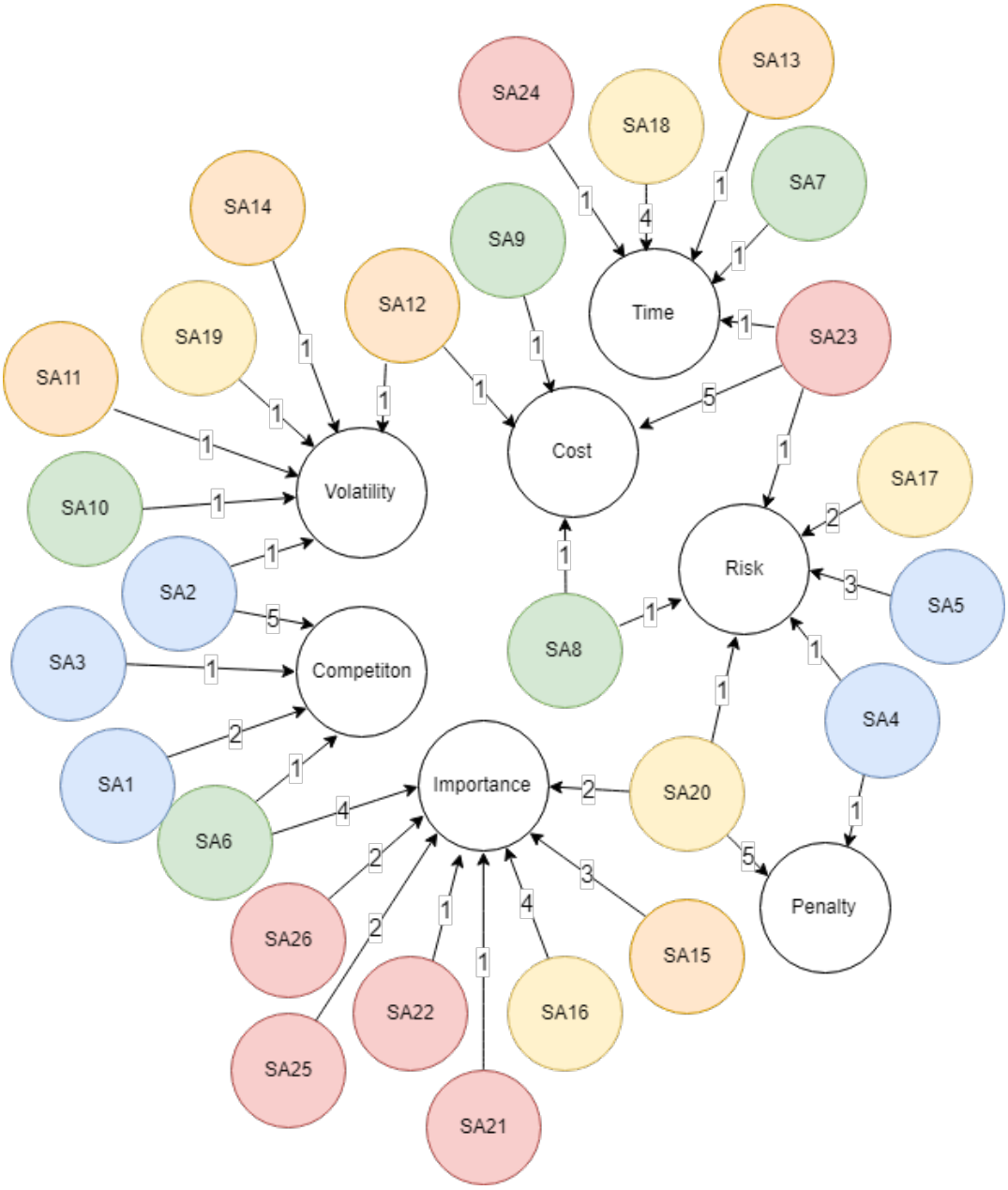

Figure 5.1: A mapping of all identified sub aspects from the first interview cycle. Each sub aspect is mapped to which generalised prioritisation aspect it was associated with by the interviewee. The numbers noted on the arrows are the number of interviewees that mentioned the sub aspect. The sub aspects are coloured after their number: SA1-5 are blue, SA6-10 are green, SA11-15 are orange, SA16-20 are yellow, and SA21-26 are red.

A1 Balance Community Edition vs Enterprise Edition

SA1 Balance Community Edition vs Enterprise Edition

A requirement that helps balance the Community and Enterprise Editions brings value to the product. I1 explained that this aspect has two sides: Both that the Enterprise Edition must be worth paying for, but that the Community Edition must still be a functional and viable option.

A2 Competitor's Features

SA2 Competitor's Features

If a competitor has an appreciated feature, it can bring a lot of value to add this as a requirement for the own product.

SA3 Market Barrier

If a requirement is considered a market standard, it can bring value to the product if implemented. I4 describes that these requirements can for example be elicited for new projects by discussing issues with the stakeholder group Experts.

A3 Complexity of Requirement

SA4 Risk of Introducing Security Holes

A requirement that risks introducing security holes to the product is time-consuming to implement and maintain. This is a potential risk and costs for the company.

SA5 Complexity

A complex requirement risks missing important details when described, which can cause problems during implementation and release. The requirement can be time-consuming to work with and add extra costs to the product.

A4 Customer Request

SA6 Source of Requirement

Implementing requirements requested by important customers can improve the relationship with the customer. This can bring extra value to the company and product. I3 added that this could be a complex issue: One customer can speak for many in cases where many have noted an issue, but not everyone communicates their opinion. Regarding the Open Source Community, I3 noted that while a single member's opinions can be hard to take into account, they become much more important in aggregates.

SA7 Urgency of Customer's Need

If a customer's need for a requirement is urgent, it is possible that the requirement will bring more value to the product the sooner it is implemented.

SA8 Customer Satisfaction

Customer satisfaction increases the probability that the customer will continue using the product for their business. This brings a stable income to the company.

A5 Developer Satisfaction

SA9 Developer Satisfaction
A feature that the developers think is useful and interesting to develop adds value to the company. I5 emphasises that developers' satisfaction with their work is a vital aspect in creating a successful company and product.

A6 Interfacing Others

SA10 Interfacing Other Teams
A higher degree of dependency on other teams' code will add the complexity of the work of the own team. This is a potential risk and cost for the requirement.

SA11 Interfacing Third Party
A higher degree of dependency on third party software will add the complexity of the work of implementing the requirement. This is a potential risk and cost for the requirement.

SA12 Syncing Teams
Requirements that require several teams to work in parallel requires coordination between teams and can increase the cost of implementation.

SA13 Training Developers
New developers need training for months before they can work on their own. Starting new, bigger requirements and projects can require developers to experiment and train to understand the problem before they can start working on them.

A7 Laws

SA14 Help Customers Adhere to Laws
According to I3, the case company's product has a potential to help companies adhere to laws regarding data protection. Implementing a requirement that can help customers adhere to laws can open up new markets and customer segments.

A8 New Market Opportunity

SA15 New Market Opportunity
A new requirement that can open the product up to new uses and new markets increases the value of the product.

A9 Product Coherence

SA16 Roadmap and Product Coherence
The concept that a requirement fits the bigger picture of the product is important and adds value to the product.

SA17 Change to Existing Features
Changes in existing features might risk that customers that use and appreciate the feature lose their interest in the product.

A10 Release Cycle Perspective

SA18 Lead Time
Time it takes to get the feature out to the product. Depends not only on implementation time but also on how it fits with releases and development cycles.

SA19 Long Term vs Short Term
The idea that some requirements are valuable in the short term and can generate revenue quickly, while others are valuable in the long term to ensure the survivability of the product and keep the code up to date. A balance between these time perspectives adds value to the product.

A11 Risk of Losing Customer

SA20 Risk of Losing Customer
The risk of losing a paying customer if a requirement is not implemented is considered a cost. Every interviewee in the first round of interviews brought up this as one of the most important aspects that is discussed within the company. I6 described the company's view on the aspects: "Ultimately, we are building software for people to use. If people stop using it, due to it not providing functionality they need, why are we building it?".

A12 Robustness and Correctness

SA21 Robustness and Correctness
A requirement that aims to increase robustness and correctness of the product adds value to the product. A very central quality characteristic since the case company's main product is a database.

A13 Showstopper

SA22 Stopping Other Teams
If a requirement has dependencies that risk stopping another team's work if not implemented soon enough, it will add cost to the product. I8 noted that from their experience, it is more likely that a small requirement will be a showstopper than a big and complex requirement.

A14 Time to Implement

SA23 Time to Implement
The amount of time that developers will need to spend on implementing the requirement is an important cost to consider.

SA24 Time to Adapt Code Base
The amount of time that developers will need to adapt the existing code base to the new feature is an important cost to consider. This can also include the need to investigate if there even is anything that needs to be adapted.

A15 Usability

SA25 Fix User Pain Point
If a lot of problems are noticed or a feature receives many complaints, a requirement that can fix this user pain point brings value to the product. User pain points are identified through both feedback from paying customers and reports from the OS community. Both I7 and I9 described that pain points that were not fixed on time often evolved into a risk of losing customers.

SA26 Improves Ease of Use
A requirement that would improve the ease of use and customer experience brings value to the product.

5.2 Measuring aspects

For each aspect, measurements have been identified in the form of questions that can be answered by the stakeholders of the requirements engineering process. The questions are taken from discussions during both the aspects interviews and the ranking interviews. In order to ensure that different requirements can be easily compared, all questions are formulated as close-ended questions with either Yes/No answers or a limited number of answers to pick from. The measurements for A4 and A14 were suggested by interviewees as they were already used within the organisation when discussing these aspects.

A1 Balance Community vs Enterprise

- Will this make Enterprise a better option for big companies? - *Yes/No*
- Will this ensure that Community is a viable option? *Yes/No*

A2 Competitor's Features

- Is this feature an appreciated feature already implemented in a competing database? *Yes/No*

A3 Complexity of Requirement

- How big is the risk of not understanding the scope of the feature? *High/Medium/Low*
- How big is the risk of missing requirements in the design phase? *High/Medium/Low*
- How big is the risk of introducing security holes in the product? *High/Medium/Low*

A4 Customer Request

- Is the feature requested by one or more of these customer groups? *Enterprise/Supported Startup/Group of Community Users*
- How urgent is the request? *High/Medium/Low*

A5 Developer Satisfaction

- Do the developers want to work on this? *Yes/No*

A6 Interfacing Others

- Which degree of regular syncing with other teams does implementation require? *High/Medium/Low*
- Will the implementation interface with other team's code? *High/Medium/Low*
- Will the implementation interface with third party software? *High/Medium/Low*

A7 Laws

- Will this help our customers adhere to a law? *Yes/No*

A8 New Market Opportunity

- Will this open up a new market opportunity? *Yes/No*

A9 Product Coherence

- Does this fit the product coherence? *Yes/No*
- Will this change any functionality that is not a user pain point? *Yes/No*

A10 Release Cycle Perspective

- Will this feature be finished within this release cycle? *Yes/No*

A11 Risk of Losing Customer

- Do we risk losing an important customer if this is not implemented? *Yes/No*

A12 Robustness and Correctness

- Does this improve robustness of the product? *Yes/No*
- Does this improve correctness of the product? *Yes/No*

A13 Showstopper

- Is this blocking other teams from working on current tasks? *Yes/No*
- Is this blocking our own team from working on current tasks? *Yes/No*
- Is this blocking an important sale? *Yes/No*

A14 Time to Implement

- How much effort is needed to adapt code base? *Little/Medium/Much/Very Much*
- How much effort is needed for implementation? *Little/Medium/Much/Very Much*

A15 Usability

- Does this improve ease of use? *Yes/No*
- Will this fix a user pain point? *Yes/No*

Table 5.1: The "Total Mentions" rank is based on how many times a certain aspect was mentioned in the aspect interviews, with rank 1 being the most mentioned and rank 16 being the least mentioned. The "Ranking" rank is based on the average score in the ranking interviews, with rank 1 being the most meaningful and rank 16 being the least meaningful. The weighted ranking is 25 % "Total Mentions" rank and 75 % "Ranking" rank. The table is sorted on Weighted Rank.

Aspect	Rank Total Mentions	Rank Ranking	Weighted Rank
A11. Risk of Losing Customer	1	1	1
A4. Customer Request	1	3	2.5
A12. Robustness and Correctness	12	2	4.5
A9. Roadmap and Product Coherence	5	5	5
A13. Showstopper	12	3	5.25
A14. Time to Implement	1	7	5.5
A15. Usability	9	6	6.75
A7. Laws	12	7	8.25
A3. Complexity of Requirement	6	9	8.25
A8. New Market Opportunity	10	9	9.25
A10. Release Cycle Perspective	6	12	10.5
A1. Balance Community vs Enterprise	11	11	11
A6. Interfacing Others	6	13	11.25
A2. Competitors' Features	4	15	12.25
A5. Developer Satisfaction	12	14	13.5

5.3 Ranking

Interviewees were asked to rank aspects from "Very meaningful" to "Unimportant" in requirements prioritisation. Table 5.1 shows how aspects were ranked, how much they were mentioned, and the weighted rank.

Two interviewees chose to not rank *Interfacing Others* with the motivation that it did not affect them in their work and they did not feel that they could rank how much it affected others. One interviewee chose not to rank *New Market Opportunity* and one interviewee chose not to rank *Showstoppers*, with the motivation that while it was important, they thought it should be handled separately from requirements prioritisation. The missing rankings have been handled as if the interviewees ranked them as "Unimportant".

Risk of Losing Customer was both one of the most frequently mentioned and the highest ranked aspect. However, I3 noted that it also depends on the importance of the customer. The potential of losing one single user does not have the same impact as if a whole company decides to stop using the product. I3 also cautioned against letting a *Customer Request* for a feature have too much importance. "Just because a customer asks for something doesn't mean that we jump into it directly to implement that thing", they said. Customer requests need to be analysed to understand the need behind the request and to use the company's expertise in their product to create a solution.

While it was not frequently mentioned during aspect interviews, *Robustness and Correctness* was ranked as the second most important aspect in the ranking interviews. I8 explained that "This is very important since our product is a database - it really has to be correct".

Despite being frequently mentioned, *Competitors' Features* received a very low ranking from interviewees. All interviewees agreed that it was more beneficial for the company to aim for being market leading by analysing customers' needs rather than relying on following competitors' products to make long-term plans for their own product.

Developer Satisfaction had a low ranking in both Total Mentions and Ranking. Both I6 and I10 thought of it as unimportant since developers get paid for their work and that a personal interest in the feature being implemented should be viewed as a bonus. Most of the interviewees agreed with this. I1 and I2, however, pointed out that people are more productive when working on something they appreciate, and chose to rank it as meaningful.

The aspect of the *Release Cycle Perspective* was frequently mentioned but did not receive a high ranking in the ranking interviews. I2 explained that this perspective has previously been an important aspect in the prioritisation, which might be why it was mentioned by several interviewees, but that today it is viewed as a secondary aspect to consider.

5.4 Summary

This chapter has presented the identified aspects that are discussed in the case company's requirements prioritisation process. A total of 26 sub aspects were identified during interviews. To make the number of aspects more manageable, sub aspects that described similar themes were grouped into aspects. A total of 15 aspects were identified from the sub aspects. All sub aspects and aspects are described in detail in section 5.1. Aspects were further analysed and 1-3 questions per aspect were formulated to measure the aspects. The questions were based on both the description of the sub aspects and further insight from when aspects were discussed for ranking. The measurements are described in section 5.2. The last step was to let interviewees rank the aspects after how important they perceived them to be in the requirements prioritisation process. Rankings for the interviewees were weighted against how many interviewees had mentioned them during the identification

process. The ranking showed that the total mentions of an aspect did not always match its perceived importance. Detailed descriptions of the rankings are described in section 5.3. A summary of all the identified aspects and their ranking is shown in Table 5.2.

Table 5.2: All identified aspects to be discussed in requirements prioritisation. The aspects are sorted after importance, with the most important being listed first.

A11.	Risk of Losing Customer
A4.	Customer Request
A12.	Robustness and Correctness
A9.	Roadmap and Product Coherence
A13.	Showstopper
A14.	Time to Implement
A15.	Usability
A7.	Laws
A3.	Complexity of Requirement
A8.	New Market Opportunity
A10.	Release Cycle Perspective
A1.	Balance Community vs Enterprise
A6.	Interfacing Others
A2.	Competitors' Features
A5.	Developer Satisfaction

Chapter 6

Prioritisation Model

The model is a summary of identified aspects that should be taken into account when analysing and prioritising requirements. The wide range of aspects identified during the aspect interviews gives a better understanding of what needs to be discussed to understand a product's requirements than simply weighing customers' needs against programming hours. This chapter will describe the model detail and a proposal on how it can be used to evaluate product requirements.

6.1 From Aspects to Model

After analysing the results described in Chapter 5, a proposal for a prioritisation model has been formulated. The changes made will be further described in this section.

Based on the overall ranking of aspects in Table 5.1, two aspects have not been included in the model: *Competitor's Features*, and *Developer Satisfaction*. These aspects received low final rankings and interviewees stated that they were considered unimportant in the prioritisation process.

The prioritisation model is no longer divided into the aspects. Instead, the measurements are divided into new overall themes that are based on which aspect they originally belonged to. First, all aspects were divided into either Values or Costs. Second, the aspects were divided into discussion topics: *Customers' Needs*, *Product Quality*, or *Timeline* for Values and *Complexity* or *Implementation* for Costs. The measurements from Chapter 5.2 were added to the discussion topics and the old aspect names were removed.

All measurements were reformulated to be answered on a ratio scale of 0-4 where 0 is Low and 4 is High. The only exception is the questions on important customer groups since these groups are absolute, not on a scale. A ratio scale has the capacity to show not just the order of prioritisation, but also how much more important a requirement is than another [3]. This is a more powerful tool than ordinal scales and will give the decision makers more information to base their prioritisation on. An absolute scale would have been even more powerful to implement. However, few of the measurements in the model could have been answered on an absolute scale with meaningful and correct estimates. It would for example be very hard to say exactly how many new market opportunities a requirement can open up, or put an absolute number on how many user pain points a requirement could fix.

The scale of 0-4 was chosen with the usability of the model in mind. A scale with more steps such as 0-9 would give decision makers more options when ranking requirements. However, findings by Lehtola et al. indicate that teams often have very different understanding of what different steps in scales signify [24]. Using fewer steps in the scale simplifies the process of gaining a common understanding of what each step should signify. This can hopefully also make the process of prioritising requirements with the model less time consuming.

6.2 Model

Values	Low				High
Customers' Needs					
Is the feature requested by one or more of these customer groups?	Enterprise 4		Supported Startup 4		Group of Community Users 4
Which degree of urgency does the request have?	0	1	2	3	4
To what degree do we risk losing an important customer if this is not implemented?	0	1	2	3	4
To what degree will this help customers adhere to laws?	0	1	2	3	4
To what degree is this blocking an important sale?	0	1	2	3	4
To what degree will open up a new market opportunity?	0	1	2	3	4
To what degree will this make Enterprise an attractive option for big companies?	0	1	2	3	4
To what degree will this ensure that Community is a viable option?	0	1	2	3	4
Product Quality					
To what degree will this improve robustness of the product?	0	1	2	3	4
To what degree will this improve correctness of the product?	0	1	2	3	4
To what degree will this improve ease of use?	0	1	2	3	4
To what degree will this fix a user pain point?	0	1	2	3	4
To what degree will this fit the product coherence?	0	1	2	3	4
To what degree Will this keep functionalities that are not a user pain point intact?	0	1	2	3	4

Timeline					
To what degree is this blocking other teams from working on current tasks?	0	1	2	3	4
To what degree is this blocking our own team from working on current tasks?	0	1	2	3	4
To what degree will this feature be finished within this release cycle?	0	1	2	3	4

Costs	**Low**				**High**
Complexity					
Which degree of regular syncing with other teams does implementation require?	0	1	2	3	4
To which degree will the implementation interface with other teams' code?	0	1	2	3	4
To which degree will the implementation interface with third party software?	0	1	2	3	4
How big is the risk of not understanding the scope of the feature?	0	1	2	3	4
How big is the risk of missing requirements in the design phase?	0	1	2	3	4
How big is the risk of introducing security holes in the product?	0	1	2	3	4
Implementation					
How much effort is needed to adapt the existing code base?	0	1	2	3	4
How much effort is needed for implementation?	0	1	2	3	4

The model is presented in this section as a set of questions with standardised answers in order to evaluate a software requirement based on the case company's values.

6.3 Priority Score

To translate the results of a model to a set of prioritised requirements, decision makers should look at the ratio between values and costs that a requirement provides to the product. The results are on a ratio scale, showing both the relative priorities of the requirements and how much more important one requirement is than another [3].

The priority of the requirements are determined by using a Cost-Value such as proposed by Karlsson and Ryan [18]. For each requirement, it must be determined how much it contributes to the total value and cost of all requirements:

$$Value(\%) = \frac{Value}{\sum Value\ of\ all\ requirements}$$

$$Cost(\%) = \frac{Cost}{\sum Cost\ of\ all\ requirements}$$

These values are plotted in a Cost-Value diagram as seen in Figure 6.1. The closer a requirement is to the upper left corner of the diagram, the more value it provides for its cost. Karlsson and Ryan suggests dividing the diagram into different areas depending on priority, where a value-cost ratio of 0-0,5 signifies low priority, 0,5-2 signifies medium priority, and 2+ signifies high priority [18]. An organisation implementing the model presented in this thesis should determine their own division of the Cost-Value diagram in order to determine the priority of their requirements. It could also be possible to not use any categories and instead analyse the diagram as a whole.

A mapping of the requirements in this sense does not produce a straightforward, ranked list of priorities like simply calculating the value-cost ratio to sort the requirements. Instead it gives the decision makers an overview of both how much of the total values and costs a requirement provides as well as the ratio. This will help to distinguish requirements with high values and costs from requirements with the similar value-cost ratio that provide much less value and costs the product. The model is not mathematically rigorous and should be treated as a guideline rather than rule, which makes the added information of the Cost-Value diagram a more appropriate view of priorities than a sorted list of value-cost ratios.

6.4 Proposed Usage of Model

The formulated prioritisation model could be used on several different levels of abstraction. The usage would depend on the requirements engineering practices of the organisation using the model and how prioritisation is carried out. This section will describe three different proposals for usage of the model: *Communication of Values*, *Basis of Discussion*,

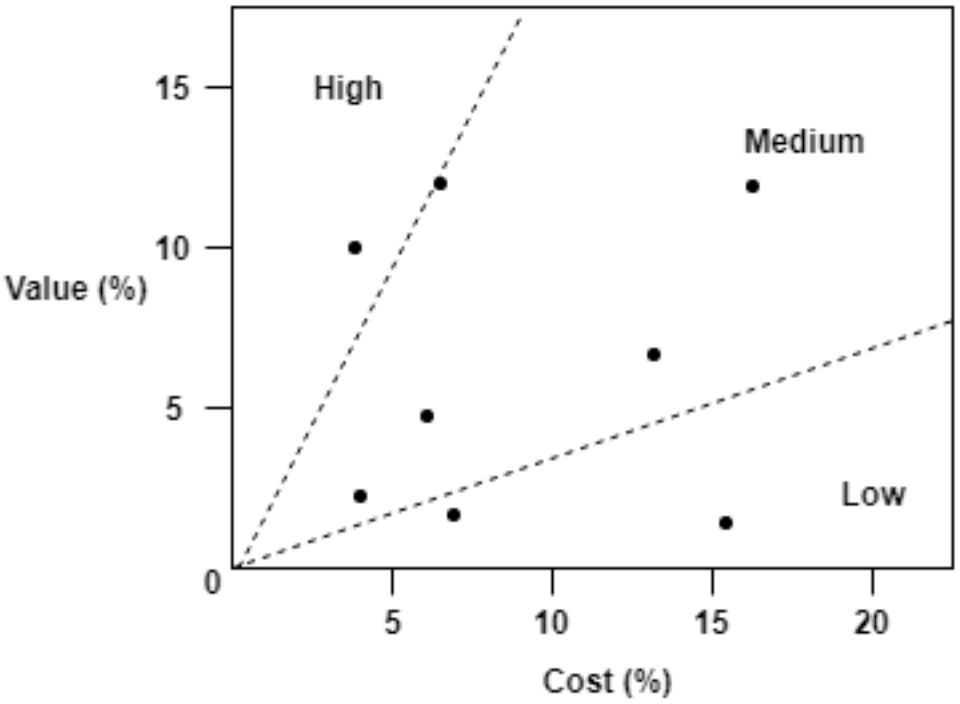

Figure 6.1: An example of a Cost-Value diagram as seen in [18]. The black dots in the diagram are example of how a plotted requirement would look.

and *Proof of Prioritisation*. An organisation could apply one or more of these usages of the model.

6.4.1 Communication of Values

If an organisation carries out requirements prioritisation on several different levels within the organisation, there will be separate groups of actors involved in the process. All prioritisation should be made based on a common set of values to reflect the overall goals and purposes of the software being created. This model can serve as the guidelines for requirements prioritisation to ensure that all actors are informed on the company values. The different actors can pick how the model should be applied based on the level of abstraction of the requirements to be prioritised. Since the same questions are used to analyse and prioritise requirements independent of their abstraction, the same values will influence the prioritisation throughout the organisation.

The model can be used to inform actors within an organisation that are currently not involved in requirements prioritisation and invite them to participate. By communicating how the requirements are currently analysed, the process becomes more transparent and

new actors can participate in discussions by giving relevant opinions that will have value in the prioritisation process.

For further explanation of this proposed scenario, see this fictitious scenario of how it could be used in an organisation:

> Recently, upper management had decided that all of the company's development teams should be agile. These changes had moved some of the responsibility of release planning and requirements prioritisation from product management to the development teams. Product Manager A had invited the team leaders to a meeting to discuss how the teams should work with requirements prioritisation. "This is a summary of the topics we in product management have discussed when we worked with requirements prioritisation," A explained as they handed out papers with the prioritisation model. "We use this to ensure that we take all sides into account when we analyse requirements." "Do we really need to fill out this check-list for each requirement?" a rather sceptical looking team leader asked. "No, no," the product manager answered. "View this as more of a reminder. Make sure to discuss all themes, or pick the ones that are most relevant. In the beginning you can use the questions as inspiration when you discuss within the team or discuss with customers. Once you get a feeling for it, just use your gut feeling and keep the model as a reminder of the core values of our product."

This usage of the model could work in all organisations that approach requirements engineering in an agile manner and treat prioritisation as a central activity that provides much value to the software development process. It will provide transparency and allow all levels of the organisation to participate, while still ensuring that the central values of the organisation are always represented in prioritisation discussions.

6.4.2 Basis of Discussion

When requirements prioritisation is done, the model can be used as a basis of the discussion. The participants can choose to discuss each question listed in the model or together decide on which questions are the most relevant to discuss for the specific requirement. The amount of questions discussed should depend both on how much time is available for the prioritisation process, and how well understood the requirement is. Free discussions of the values and costs of the requirements will help with analysing the requirement and deepening all participants' understanding of its priority, while still ensuring that the organisation's values are central to the discussion.

For further explanation of this proposed scenario, see this fictitious scenario of how it could be used in an organisation:

> Product Management had gathered for prioritising the software requirements

> of their next release. They had agreed that the main feature of the release should be customer specific profiles, and now it was time to prioritise the requirements of the feature. "I think that it is important that delivery performance prediction is implemented soon," stated A. "We have several customers that requested it and I've heard from Sales that us not having AI for predictions is blocking several sales." "Yes, it is important, but look at its complexity - we know so little about this and we risk missing requirements." B held up the paper with the prioritisation model and pointed to the different questions as they continued explaining. "It will require us to use third party software and will need several of our teams to collaborate. I think that it is a high cost for not enough value. Shouldn't we prioritise visualisation of performance data instead? It is highly requested, and it will bring much value through the product quality. I talked to the developers and they thought that it could improve the usability of our software. It is not nearly as complex and we get much more value from the cost." "Okay, fair point, we should probably prioritise the data visualisation over predictions," A agreed.

This usage of the model could work in organisations where all or most decision makers in the prioritisation process are able to participate in meetings. Participants can discuss requirements and together make an estimate on how the requirements should be prioritised without having to document the exact priority scores. This can be a beneficial approach when there are large amounts of requirements discussed at the same occasion. If all questions in the model are to be discussed in enough detail to be scored, the meetings risk being too time consuming.

6.4.3 Proof of Prioritisation

When the decision makers in the prioritisation process are not able to actively participate in the detailed discussions of priorities, the model can be used as a proof to the decision makers that the results of the discussions yield the most beneficial priorities. The actors who have knowledge on the requirements can discuss them and rank them using the priority scores. The resulting list of prioritised requirements can be used to inform the decision makers of the results of the discussions and to gain the approval to continue the software development.

For further explanation of this proposed scenario, see this fictitious scenario of how it could be used in an organisation:

> The executives had asked for a proposal on how they should prioritise the requirements in the next release, and a motivation for why this prioritisation would benefit the company. Before the meeting, the product managers had already split up which questions each one should investigate for the prioritisation meeting, and now they had gathered to summarise their findings.

> "Let's go from the top and start with 'Prediction algorithms for delivery performance'," secretary A said. "B, you have talked to sales and marketing, how does the customers' needs look?" "Many important customers have requested it. However, we agreed that it is not very urgent and nothing we risk losing customers over, and neither did we find any legal benefits for the customers. Sales are however pretty sure that they can close a few sales if we implement it, and marketing think its a good opportunity to market us in a new customer segment." "That's great," A responded. "C, did you talk to the community managers about it?" "I did," C answered, "and honestly, they are not happy that we only want to provide this to the paying customers." "That's to be expected," said A. "I guess I'll put that as a "0" for making Community a viable option?"

This usage of the model could work in an organisation where the decision making process is centralised. The decision makers can delegate the responsibility to discuss requirements to actors that have a deep understanding of the requirements and still ensure that the organisation's values are being followed.

6.5 Evaluation of Model

Interviewees I3 and I4 have roles that are central to the current requirements engineering and prioritisation practices and are both decision makers for high level requirements prioritisation. An evaluation of the proposed prioritisation model was conducted through validation meetings with I3 and I4 separately. Validation was done by going through the proposed model and then discussing the proposed usages.

Both I3 and I4 considered the overall themes in the model very relevant to the requirements prioritisation of the case company. The questions to measure the themes were recognised by both as considerations that are brought up in the analysis of requirements. As a whole, both interviewees agreed that the model was a good reflection of how the case company aims to analyse and prioritise their product requirements.

In regards of the proposed usages of the model, both interviewees considered *Basis of Discussion* the most relevant usage for the case company. It fits well with their current practices of working with people and discussions rather than documentation. Since the model has many questions to answer and can be complex to use, it would be very time-consuming to use it as a *Proof of Prioritisation*. I3 also noted that since it is almost always possible to have all decision makers of the requirements engineering process participating in prioritisation discussions, there is no need for further proof that the decided prioritisation is the most beneficial. When asked if any themes or questions would be considered more important and relevant than others, I3 stated that this would likely differ between requirements. Having all themes and questions in the proposed model available could be a tool to help see all aspects that potentially need to be analysed and let the decision makers pick which ones are most relevant to discuss thoroughly.

I4 stated that it could be interesting to use the model for *Communication of Values* in order to open up discussions with Product Engineering on prioritisation on both high- and low-level requirements. Again, the potential complexity of the proposed model makes it time consuming to use, and *Basis of Discussion* would likely be too time-consuming for prioritisation of low-level requirements. They noted that the model would most likely be more useful as a way of reminding all actors to widen discussions to take all important values into account.

6.5.1 Comparison to Similar Model

Wiegers presents a semi-quantitative analytical model to analyse and prioritise product requirements [30]. The model is described in eight steps:

1. List all requirements, features or use cases. Ensure that all are described on the same level of abstraction.
2. Estimate the benefit for the customer or the business for each requirement. Use a scale of 1-9 where 1 is very little benefit and 9 is the maximal possible benefit.
3. Estimate the penalty that the customer or the business would suffer if the feature is not included. Use a scale of 1-9 where 1 is no penalty and 9 is very serious penalties.
4. Calculate the Total Value for each requirement as the sum of the benefit and the penalty.
5. Estimate the cost of implementing each requirement. Use a scale of 1-9 where 1 is very low cost and 9 is very high cost.
6. Developers estimate the degree of risk in implementing each requirement. Use a scale of 1-9 where 1 is being able to program it in your sleep and 9 is serious concerns for the risk not having the tools or resources to implement the requirement.
7. For each requirement, calculate the following values:
   - Value%: Divide the Total Value of each requirement with the sum of Total Value for all requirements.
   - Cost%: Divide the Cost of each requirement with the sum of Cost for all requirements.
   - Risk%: Divide the Risk of each requirement with the sum of Risk for all requirements.

   Priority is then calculated for each requirement as:

$$Priority = \frac{Value\%}{Cost\% * CostWeight + Risk\% * RiskWeight}$$

   where Cost Weight and Risk Weight can be used if Value, Cost and Risk is not to be weighted equally.

8. Sort the list in descending order by priority. The features at the top of the list should have higher implementation priority.

Wieger's model uses the four themes *Benefit*, *Penalty*, *Cost*, and *Risk* while the model proposed in this thesis uses *Customer's Needs*, *Product Quality*, *Timeline*, *Complexity*, and *Implementation*. While there are no detailed description of what Wieger's themes entail, the descriptions of *Benefit* and *Penalty* overlap somewhat with *Customer's Needs* and *Product Quality* while *Cost* and *Risk* overlap somewhat with *Complexity* and *Implementation*. Applying Wieger's model relies heavily on the people involved being well aware of the values of the organisation in order to estimate the values in the model. The model proposed in this thesis solves this issue by providing discussion topics in the form of questions. While at can be argued that Wieger's model can be applied to organisations with different values, the questions in this thesis are meant to be generalised enough to fit a wide range of organisations.

It is pointed out that Wieger's model is not mathematically rigorous and should be used as a guideline rather than a rule [30]. The same argument applies to the model in this thesis. Wieger's model applies the prioritisation scores to the themes and uses a scale with more steps, while this proposed model assigns scores to each questions and uses a smaller scale. It can be argued that having to assign more scores is more time consuming and reduces the usability of the model. However, an organisation wishing to apply Wieger's model must identify their own guidelines for measuring the themes without any support equivalent to the questions in this model. It should also be noted that a scale with more steps might not be more precise than fewer steps, as humans struggle with using absolute scales consistently [24] [26].

The model proposed in this thesis seems to have a slightly different purpose than the model proposed by Wiegers. Wieger's model is described as a "...semi-quantitative analytical approach to requirements prioritisation" and the author urges organisations to use the model as a base that they must adjust to fit their own needs [30]. It is a suggestion of a simplified approach meant as an alternative for more time consuming techniques. The model proposed in this thesis is on the other hand meant as a way of broadening the concepts of Values and Costs. The most important part of this model is the diverse set of questions that should be discussed for a bigger understanding of concepts central to requirements prioritisation. Instead of proposing an adaptable approach, this model focuses much more on the contents of the requirements analysis.

6.5.2 Countering of Identified Challenges

Table 6.1 summarises whether the proposed prioritisation has handled the challenges presented in 4.1 or not.

Both *C1. No Guidelines* and *C2. Hard to Communicate* are handled well by the model since the proposed prioritisation model is a set of questions meant to thoroughly analyse requirements. It is an easy way to align the values within the company to ensure that

everyone knows what is important when analysing requirements. It is also easy to communicate these values by simply keeping the model available for anyone who has a need to understand the prioritisation process.

C3. Lack of Documentation and *C4. No Customer Present* were taken into account when deciding on the appropriate level of abstraction of the questions in the model. The questions do not need detailed descriptions of design and exact usage to answer, but instead focuses on high-level values and qualities. This information does not require detailed design and relies more heavily on input from Product Engineering and Product Management than from the customers and end users.

C5. Requirements Types and *C6. Many Requirements* are only partially handled and depend on how the model is implemented in the organisation. If an organisation was to apply Proof of Prioritisation, the process would be very time consuming. It is also not certain that all different kinds of requirements would receive equal weight in the model, since there are not an equal number of questions that apply to each requirement type. It would probably be required to evaluate different requirement types separately. If the organisation applies Basis of Discussion or Communication of Values, the model does handle these challenges. Organisations can pick how much time is to be spent on prioritisation with the model, and will get a wide range of values and costs where they can pick which questions are the most important for each requirement depending on its type.

Table 6.1: A summary of the identified challenges that the model should handle, and a conclusion if they were handled or not.

ID	Description	Handled
C1. No Guidelines	Since there are no concrete guidelines on how prioritisation should be made, it will differ depending on who is involved in the process.	Yes
C2. Hard to Communicate	Values for prioritisation are hard to communicate to new employees or employees that have previously not been involved in the prioritisation process.	Yes
C3. Lack of Documentation	The case company does not rely on documentation. Requirements that are to be prioritised are high-level and not thoroughly documented and designed.	Yes
C4. No Customer Present	No customer and end-user can actively participate in the requirements engineering process and the model cannot rely on extensive customer feedback.	Yes
C5. Requirements Types	There are many different kinds of requirements types that need to be handled in the requirements prioritisation process.	Partially
C6. Many Requirements	Each time requirements are prioritised, there are a big amount of requirements to analyse. The prioritisation process cannot be too time consuming for each requirement.	Partially

Chapter 7

Threats to Validity

Several threats to the validity and reliability of the results presented in this report have been identified and countered. To give the reader an understanding of these threats and their effect on how results should be interpreted, this chapter will discuss these threats and counter measures. The threats are divided into four categories as described by Runeson et al [27]: *Internal Validity*, *External Validity*, *Construct Validity*, and *Reliability*.

7.1 Internal Validity

The internal validity of a study is threatened when unknown factors affect the results when studying causal relationships [27]. In this case study, the biggest threat to the internal validity is that the work history of the interviewees that is unknown to the researcher. The discussions on prioritisation aspects are very likely to be affected by what has been discussed in the most recent projects that an interviewee has worked on. Since the researcher has no insight into the requirements engineering of previous projects, their impact on answers during the interviews is unknown. There is a risk that the researcher has troubles separating which aspects are emphasised as important due to being fresh in memory, and which are important due to affecting many different requirements and projects throughout the organisation.

To counter this threat on the interval validity, interviewees have been chosen from a wide range of teams and roles within the company. Even if the history is unknown, it is unlikely that all interviewees have recently participated in the same project. Aspects that are

discussed by several interviewees are likely to be important throughout the organisation and not just for one recent project. The initial stakeholder analysis helped with picking interviewees that were unlikely to have been affected by the same work history.

7.2 External Validity

The external validity of a study refers to its ability to be generalised and to which degree the results can be of interest for actors outside of the case study [27]. There are two generalisations that this study aims for: First, the ability to generalise the findings throughout the case company. Second, the ability to generalise the findings to be usable for similar companies.

The ability to generalise within the company is threatened by the fact that only a small part of all employees could be interviewed as part of the case study. To ensure that the findings are as representative of the whole company as possible, interviewees have been chosen from several teams and roles. It has also been ensured that Product Engineering, Product Management, and Executives have all been represented in both interview cycles so that all three perspectives are taken into account in the results.

Since the purpose of this report is to study general research questions through a case study, it is important that the results are applicable in a wider context than the case company. The chosen case company is in its size, domain, and software development methodologies similar to many other companies, which is important for the generalisability of the results. A threat to this ability is the risk of focusing too much on aspects related to specific projects at the case company rather than bigger company values. This threat has been handled by explicitly asking interviewees to discuss themes that are reoccurring in several projects.

7.3 Construct Validity

Construct validity is considered to ensure that what is being investigated in a case study is the same question that the researcher intended to investigate [27].

Since only one researcher conducted the case study, there is a risk that unidentified researcher expectations have influenced the results of the study. To counter this, all research questions have been kept open to avoid encouraging interviewees to give certain answers. All results have continuously been validated together with interviewees to ensure that their opinions are correctly reflected and that the results correctly reflect the real company values.

Another risk is hypothesis guessing; that participants in the study might think they know what the researcher wishes to hear and answer accordingly. This has been countered mainly by having a large number of interviewees and weighting all opinions as equally

important. If one or a few interviewees have tried giving the answers they think that the researcher wishes to hear, it has had little effect on the results. Having open questions in the interviews have also served the purpose of making the researcher's hypothesis harder for participants to guess.

7.4 Reliability

Reliability refers to to which degree the results of a study is dependent on the researcher that conducted the case study [27].

In the aspect identification interviews, the questions were open and the interviewees led the discussion. This means that the results might have differed based on the interviewer. The words that the interviewees were asked to discuss around are described with the definitions used during the interviews to counter this threat to reliability. For the ranking interviews, the questions were not as open and another interviewer should be able to get the same results through the described methodology. The questions in the aspect identification interviews were based on general prioritisation aspects from literature. However, the ranking methodology from the ranking interviews is not based on literature and it is hard to confirm if the chosen methodology is the best way to get the correct results.

All data was collected through written notes rather than recordings and there is a risk that the interviewer left out information. To counter this, all notes were read through directly after interviews and the interviewees were asked to provide clarification if some information seemed to be missing or if the notes were not clearly understood.

During the formulation of the prioritisation model, two aspects were removed. This was done due to the aspects receiving low rankings and several interviewees stating that is should not be part of the prioritisation process. To ensure that no information is lost, the aspects were still described with the same detail as other aspects. This means that if another researcher decided to use the aspects in the model, they could simply add it by using the information from the results in Chapter 5.

Chapter 8

Discussion

This chapter is divided into two parts. First, the research questions that were the base of this study are discussed and answered. Second, more considerations to take into account when applying a prioritisation model in a real life setting is discussed.

8.1 Research questions

This section aims to discuss how the results can answer the research questions presented in the introduction of this thesis.

RQ1. Which aspects should be taken into account in agile requirements prioritisation for companies that develop open source products?

This research question is answered by the proposed prioritisation model i Chapter 6. The interviews show that all interviewees have a wide range of aspects that they consider important to ensure that the product is of high quality, and think that these should be taken into account when analysing and prioritising requirements. The final overall themes to describe the aspects in the model are *Customers' Needs*, *Product Quality*, *Timeline*, *Complexity*, and *Implementation*. The proposed prioritisation model goes against the agile requirements engineering principle of prioritising according to customers' needs and es-

timating costs throughout the implementation. Deviating from the principle is supported in other studies where it has been found that that pressure to only deliver what customers request hurts the quality of the product [5].

It should be noted that as described by Inayat et al., agile methodologies assume that the customer making the request is actively participating the the development process and has much knowledge of the product [17]. This is not how the case company interacts with their customers. Instead, there are many customers within different domains spread over the world, which makes it much like market driven development. Customers of the case company are seldom knowledgeable enough about the code base and the core of the product to request changes that will improve for example robustness, correctness, or other important quality indicators of the product. The case company also faces the issues with volatile requirements that are typical for market driven development. Changing requirements will mean that a lot of time will be spent on analysing requirements. The more factors that need to be taken into account, the more resources will be spent on analyses. There will be a point where the organisation must weight the benefits of thorough analysis against the cost of conducting it. On some projects, where requirements are volatile, analysing what customers need and estimating implementation time might be more fitting than analysing more factors. However, on most projects, the results from the interviews suggest that much important information will be lost if more aspects are not taken into account.

The open source aspect of RQ1 has had a different impact than initially speculated at the case company. It was at first assumed that requirements would be analysed differently depending on source, but the main difference appear to actually be the elicitation phase. Paying customers are much more integrated in the software development process and have an easier time communicating requirements. The Community Users have much more limited contact with the decision makers of the company which affects their ability to give input. However, once requirements are elicited, interviewees state that they are all analysed the same way independently of source. The only impact of source is that perceived value increases when an important customer has requested it. In the case of open source users, they are considered more important if a big number of community users stand behind the request. If the proposed prioritisation model were to be applied in an company with a open source community which has more influence on the requirements engineering process, there might be a need to add more aspects to the model that is currently not considered at the case company. This could be aspects that, for example, handle giving the community more incentives to participate, or that makes community contributes easier to access.

Benefits of having an open source community can be both to receive input and to have a distribution channel [10]. Community Users of the case company's product do not have the same ability to provide input as Paying Customers, and are viewed much more as a distribution channel. This is a result of the complexity of the case company's product, which has lead to most contributions being made in the form of add-ons rather than code for the core product. However, while the Community Users might not contribute much in form of code, they can still contribute valuable opinions to the requirements engineering process. Their opinions might represent the opinions of the users at the companies that are Paying Customers, or they might express needs that are currently stopping them from

becoming Paying Customers. The case company could use the concepts of *Accessing*, *Aligning* and *Assimilating* presented by Dahlander and Magnusson if they wish to gain more input and innovation from their user community [11]. Having better routines for collecting input could help the case company to access the innovation of the community, and providing feedback from different levels within the company could be an incentive for Community Users to continue contributing. By being open about company values and the requirements prioritisation process, the case company could help align the goals of the community with their own. Assimilating the innovation could be done by using resources internally to analyse the requests and proposals from the community to ensure that good ideas are not lost due to lack of time or interest. If the Community Users could see that their requests are taken seriously or even implemented in the product, it would be an incentive to contribute more. If the community had closer contact with the case company and could learn more about the company values, hopefully the requests and proposals would increase in quality over time as the relationship evolved.

RQ2. How should these aspects be measured?

The proposed prioritisation model measures the aspects through questions with a ratio scale of 0-4. This is meant to make the model easy to apply in different organisations and different levels. More detailed guidelines for how measurements should be conducted would make the model hard to generalise since it would be based on current practices of the case company. Having less detailed guidelines or only providing the aspects without the measurements would not communicate which considerations should be discussed in the prioritisation process. It can be argued that a simple scale of 0-4 is not enough to capture the complexity of the questions presented. However, it is assumed that these questions are answered through discussions between practitioners with extensive knowledge of the product requirements rather than through using the model as a quick check-list. The practitioners can adapt how the proposed questions should be answered to fit their organisation if more complexity is needed.

As discussed in the proposed usages of the model, the level of detail of measurement should be up to the organisation to decide. Anything from only discussing the most important aspects to using a detailed scoring system should be possible when using the model. However, it is very likely in an agile organisation that the questions will be answered using estimates due to lack of detailed design of features. More considerations on this is discussed in section 8.2.

RQ3. How can this model be applied in a company setting?

Proposed usage scenarios of the prioritisation model were presented in chapter 6. The model gives the organisation a lot of freedom in applying the model to fit their current practices. Common for all usages is that the model must involve input from different actors within and outside the organisation, and must be established and understood by all of these

actors. Understanding customers' needs requires input from both customers and Product Management. To understand the technical aspects of the product and implementation, Product Engineering must the consulted. While developers are seldom decision makers of high-level requirements, it is important to involve them and their opinions on the plans for future development work. Agile development teams are much more likely to succeed with agile principles if they are given responsibility and are allowed to give input in the planning of their work [16].

Further discussions on considerations for applying the proposed prioritisation model are presented in section 8.2.

8.2 Considerations for Applying Model

Not all of the discussed considerations from the interviews could be included in the proposed prioritisation model. These considerations are on a higher level of abstraction than the aspects presented in the model, and too much information is lost if they are to be summarised as questions in the same form as the other aspects. The main issues brought up by interviewees can be split into three different categories. First, the discussion on which abstraction levels of requirements this type of model is useful for. Second, which time frame to use when analysing and prioritising requirements. Last, the issue of working with estimates and predictions, and the accuracy of these.

8.2.1 Abstraction Levels of Requirements

All proposed usages of the requirements model described in Chapter 6 can be used on software requirements of different abstraction levels. This was not unexpected, since the chosen interviewees handle requirements on different levels within the organisation and the results aimed to reflect the needs on all levels. However, the question is on which levels of abstraction the model is actually useful.

For low-level requirements, the sort of prioritisation that is done each agile iteration [25], the prioritisation process cannot be too complex or time consuming. Discussing each question in the proposed model for each requirement will take too much resources from development. The proposed usages of Communication of Values or Basis of Discussion from Chapter 6 could be useful. Development teams often take time to adjust to the responsibility that comes with agile development [4]. A team that is new to agile requirements prioritisation could benefit from the more structured way of Basis of Discussion. If agile methodologies are more established, which is the case at the case company, Communication of Values might be enough to help the teams make informed decisions. The prioritisation model might not be the most useful tool on this abstraction level, as technical dependencies are likely to determine a lot of priorities.

For more high-level requirements that might be part of a release plan or span a few iterations, there are fewer requirements and also more risk involved in failed prioritisation. The actors involved in the prioritisation process are likely to have insight in both the business aspects and technical aspects of the product. If not, they are likely to know how to get the information. They can most likely determine which aspects in the proposed prioritisation model that are useful to analyse a certain requirement. This would mean that Basis of Discussion could be an efficient and useful way of using the prioritisation model. Depending on the demands from the rest of the organisation, Proof of Prioritisation could sometimes be a necessary usage.

Very high-level requirements for long term strategies are most likely not described on a level of detail where useful estimates can be made. This means that the proposed prioritisation model is not very useful. It should also be noted that the proposed prioritisation model does not value innovation and new ideas, which is important for successful long-term planning. Using the prioritisation model as Communication of Values might have some usefulness in keeping a coherence in the product development, but the model is not created to support this kind of prioritisation.

8.2.2 Time Frames for Prioritisation

Many interviewees brought up the issue of on which time frame a software requirement should be evaluated. On one hand, it can be looked at on a short time frame, like a release cycle or less. In that case, business values or lack of business values will most likely have a high impact on its importance and value. On the other hand, if a requirement is looked at for a longer time frame, some technical values might be more prominent. This issue can be viewed as that not prioritising technical improvements of the code base will lead to a technical debt. While it will not provide new features that can be used to generate revenue and fulfil business goals, it will with time lead to issues and quality problems. In Chapter 5, this consideration was described as sub aspect SA19 which was deemed too complex to add to the model in a meaningful way.

The issue with business values versus technical needs is not unique for this company and many researchers have found that agile organisations struggle with this balance [5]. Agile development methodologies encourage a short-term perspective to ensure that customer values are delivered. But as shown in these results, the long-term risks of a technical debt is a serious risk and must be considered in the prioritisation process. It could also be argued that if not enough technical aspects are considered in the short term, business values will suffer in the long term when the technical debt affects the quality of the product and the ability to expand it with new features. The risks are bigger for high-level requirements with a big strategic value than for low-level requirements and day-to-day tasks.

How should these considerations be taken into account when applying the proposed prioritisation model? One way could be to do two separate analyses of high-level requirements. First, a short term analysis that focuses mainly on the next release cycle. Then, a long-term analysis that looks at how the requirement's implementation will affect the product in two

or three release cycles. Both analyses should then be considered when prioritising the whole set of requirements. This could open up discussions on how a technical debt will affect the long-term survivability of the product without ignoring the need to maintain a stable revenue in the short term. Another way to handle the time frames could be to track dependencies between requirements and then analyse them using the model. Tracking dependencies can be a complex issue and should involve both management and engineering to fully understand all dependencies. This could highlight dependencies between requirements that solve technical issues and those that solve business needs. For example, it could show that a implementation of an important business need can be done with a higher quality and lower cost if a technical requirement to improve the existing code base is prioritised and done first. This can be time consuming since it requires both analysis of dependencies and analysis with the model for the different scenarios. However, it can also help finding ways to reduce the complexity and time required to implement many planned features.

8.2.3 Accuracy of Estimates

All aspects in the proposed prioritisation model are to some degree uncertain and all answers must in some sense be regarded as estimates. Interviewees pointed out that some of these estimates, especially if requirements are evaluated for longer time frames as suggested in the preceding section, are very uncertain. The model tries to take this factor into account as "complexities" - the risk of not understanding or missing requirements. However, even this is an estimate as a requirement can seem less complex before you actually start working on it and designing the solution.

One solution to this could be to use prototypes and proofs of concept to analyse requirements. If an engineer can work on a high level design of the solution before the requirements are analysed and prioritised, it is much easier to estimate the answers in the model. There are two main problems with this solution: Allocating resources to prototyping instead of developing the product is expensive, and prototypes can be misleading. The first problem means that prototyping and proof of concept should only be made for requirements that have a high values and high costs, since these have the potential to be very valuable but also carry high risk if the estimates turn out to be incorrect. The second problem means that prototyping must be well thought through and analysed in order to ensure that the right parts of the feature are prototyped so that the results can be used to correctly understand the actual implementation.

An alternative solution could be applied to handle the risk of increasing volatility of estimates as the time frames increase. Instead of evaluating and prioritising the requirement before implementation starts, the prioritisation could be re evaluated during the development process. This would allow for correction of estimates to better understand the total values and costs of a requirement. Either the feature should be redesigned to better fit the new estimates, or the development plan should be changed. The main issue with this is that it will damage the throughput of the organisation. It has been found that costs are often underestimated in the requirements prioritisation process [6]. If a re-prioritisation shows that the costs of a requirement has increased while the value stays the same, the de-

velopment might be delayed in order to prioritise a more important requirement. Having half-finished features that are delayed over and over again is a waste of resources and will lead to a low throughput. Thus, re-prioritisation should be done with care.

Chapter 9

Conclusions

This masters thesis was conducted to investigate how software requirements prioritisation should be conducted in an agile setting. A case company was studied in order to formulate a prioritisation model. Since the case company works with open source, the study also included an analysis of how having both paying customers and an open source community affected the requirements engineering process. The thesis also investigated which other considerations needed to be taken into account if an organisation were to implement the proposed prioritisation model.

Interviews showed that the case company has very different structures for eliciting requirements from paying customers and the open source community. There are also several internal sources of software requirements. All requirements are eventually analysed and prioritised by either Product Management or Product Engineering. While Product Management mainly handles high-level requirements, Product Engineering handled low-level and technical requirements. The organisation relies heavily on communication between individuals and tries to keep documentation and pre determined processes at a minimum.

The proposed prioritisation model is a result of interviews where interviewees were asked to describe what they thought were important subjects to discuss in order to understand and prioritise software requirements. By analysing these results, certain reoccurring subjects were described as so called aspects. These aspects were further discussed and later ranked by interviewees in order to understand their relevance for the case company. For each aspect, measurements to measure the aspect was formulated, and this was collected into a prioritisation model. To complete the model, a system for scoring priorities and three proposed usage scenarios were formulated.

Results from interviews have shown that there are high level considerations that need to be taken into account while prioritising requirements. Not all of them could be included in the prioritisation model, and those were discussed separately. These considerations focused on how the model can be used: On which abstraction levels, in which time frames the requirements should be viewed, and the degree of accuracy of estimates. These discussions show that a lot of information can be lost if requirements are not viewed both from different levels of the company and that short term and long term importance of requirements do not always conform. There is also a risk that estimates used for answers in the model cannot be made accurately if the requirements are too abstract or if the analysis is done too far ahead.

In conclusion, this thesis shows that agile requirements prioritisation can be a complex process with many factors that need to be taken into account. These results can hopefully help clarifying important aspects to discuss in the requirements prioritisation for both the case company and other, similar companies. Both the proposed model and additional considerations presented in the discussions aim to help organisations improve both their understanding of the prioritisation process and its results.

Bibliography

[1] P Achimugu, A Selamat, R Ibrahim, and M Naz'ri Mahrin. A systematic literature review of software requirements prioritization research. 2014.

[2] K Beck. Embracing change with extreme programming. 2000.

[3] P Berander and A Andrews. Requirements priorisation. 2005.

[4] E Bjarnason, K Wnuk, and B Regnell. A case study on benefits and side-effects of agile practices in large-scale requirements engineering. 2011.

[5] N Brede Moe, A Aurum, and T Dybå. Challenges of shared decision-making: A multiple case study of agile software development. 1997.

[6] L Cao and B Ramesh. Agile requirements engineering practices: An empirical study. 2008.

[7] L Chan and M Wu. Quality function deployment: A literature review. 2002.

[8] H Chesbrough. *Open Innovation: The New Imperative for Creating and Profiting from Technology*. Harvard Business School Publishing Corporation, 2003.

[9] H Chesbrough and M Appleyard. Open innovation and strategy. 2007.

[10] S Comino and F Manenti. Dual licensing in open source markets. 2007.

[11] L Dahlander and M Magnusson. How do firms make use of open source communities? 2008.

[12] M Fowler and J Highsmith. The agile manifesto. 2001.

[13] D Greer and Y Hamon. Agile software development. 2011.

[14] J Henkel. Champions of revealing - the role of open source developers in commercial firms. 2009.

[15] A Herrmann and M Daneva. Requirements prioritization based on benefit and cost prediction: An agenda for future research. 2008.

[16] R Hoda, J Noble, and S Marshall. Developing a grounded theory to explain the practices of self-organizing agile teams. 2011.

[17] I Inayat, S Salwah Salim, S Marczak, M Daneva, and S Shamshirband. A systematic literature review on agile requirements engineering practices and challenges. 2014.

[18] J Karlsson and K Ryan. A cost-value approach for prioritizing requirements. 1997.

[19] J Karlsson, C Wohlin, and B Regnell. An evaluation of methods for prioritizing software requirements. 1998.

[20] L Karlsson, Å Dahlstedt, B Regnell, J Natt och Dag, and A Persson. Requirements engineering challenges in market-driven software development - an interview study with practitioners. 2008.

[21] L Karlsson, T Thelin, B Regnell, P Berander, and C Wohlin. Pair-wise comparisons versus planning game partitioning—experiments on requirements prioritisation techniques. 2006.

[22] K Lakhani and R Wolf. Why hackers do what they do: Understanding motivation and effort in free/open source software projects. 2005.

[23] S Lauesen. *Software Requirements - Styles and Techniques*. Pearson Education, 2002.

[24] L Lehtola, S Kujala, and M Kauppinen. Requirements prioritization challenges in practice. 2004.

[25] F Paetsch, A Eberlein, and F Maurer. Requirements engineering and agile software development. 2003.

[26] B Regnell, M Höst, J Natt och Dag, P Beremark, and T Hjelm. An industrial case study on distributed prioritisation in market-driven requirements engineering for packaged software. 2001.

[27] P Runeson, M Höst, A Rainer, and B Regnell. *Case Study Research in Software Engineering - Guidelines and Examples*. John Wiley & Sons, 2012.

[28] K Schwaber. Scrum development process. 1997.

[29] H Sharp, A Finkelstein, and G Galal. Stakeholder identification in the requirements engineering process. 1999.

[30] K Wiegers. First things first: Prioritizing requirements. 1999.

[31] L Williams and A Cockburn. Agile software development: It's about feedback and change. 2003.

EXAMENSARBETE A Model for Value and Cost Trade-offs in Agile Software Requirements Prioritisation

STUDENT Elin Blomstergren
HANDLEDARE Johan Linåker (LTH)
EXAMINATOR Björn Regnell (LTH)

Förbättrad kravprioritering i agil mjukvaruutveckling

POPULÄRVETENSKAPLIG SAMMANFATTNING **Elin Blomstergren**

Agil mjukvaruutveckling har gjort kravprioritering till en central aktivitet i utvecklingsarbetet. I detta arbete studeras hur kravprioriteringen kan förbättras när man ser bortom kundbehov och implementeringstid.

För att hantera en affärsmiljö där marknaden förändrades snabbt och ofta så föddes under 1990-talet de agila mjukvaruutvecklingsmetoderna. Metoderna förespråkar ett iterativt arbetssätt där projektplanering och utvärdering av kundbehov utförs genom hela projektets gång för att snabbt svara på förändringar. I och med detta blev kravprioritering en central aktivitet för att ständigt hålla utvecklarna uppdaterade om vad som är mest värdefullt att utveckla. Många agila metoder föreslår att i var iteration ska kunden lista de krav som är mest värdefulla för dem och utvecklarna ska uppskatta hur lång tid detta kommer ta att implementera. Men studier visar på att många projekt får problem när kundbehov styr utvecklingen.

Mitt examensarbete studerar hur begreppen nytta och kostnad kan expanderas för att bygga en modell för bättre kravprioritering. Genom en fallstudie på ett medelstort svenskt företag har jag undersökt vilka andra faktorer som måste övervägas för att fullt ut förstå vad som ökar kundnyttan hos produkten och vad som påverkar kostnaden av implementationen. Ytterligare komplexitet tillkommer av att företagets produkt är Open Source och betalande kunders önskemål måste vägas mot företagets OS communitys åsikter.

Totalt intervjuades 11 anställda på företaget vid 16 intervjutillfällen. Intervjuerna gjordes med öppna frågor där de anställda ombads berätta öppet om deras tankar om kravprioritering och produktvärde. 26 faktorer identifierades som ansågs bidra med nytta eller kostnad i utvecklingsprocessen och dessa grupperades till 15 övergripande teman. Efter input från de anställda så bedömdes 13 av dessa teman vara värdefulla i prioriteringsprocessen. En kravprioriteringsmodell formulerades genom slutna frågor för att mäta varje tema. Slutligen gjordes en analys av hur denna teoretiska modell skulle kunna implementeras i en organisation.

Slutsatsen som kan dras från detta arbete är att kravprioritering kan bli en mycket komplex process om alla viktiga faktorer ska analyseras och övervägas. Detta medför att företag måste fokusera på att bygga rutiner för att effektivisera kravprioriteringsprocessen. Rekommendationen till det studerade företaget och liknande företag är att i långsiktig planering utnyttja en prioriteringsmodell likt den föreslagen i examensarbetet för att göra en grundlig analys av både kundbehov och företagets egna behov. Det rekommenderas också att vara transparent med processen och informera de som jobbar med det löpande, iterativa prioriteringen om prioriteringsmodellen som ett stöd i deras kravanalys.