NTNU

Software Requirements Engineering

A Case Study of a Startup

Norwegian University of Science and Technology
Department of Computer Science
Supervisor: Eric Monteiro

Master of Science in Technology in Computer Science

Erlend Morthen

Submission date: 12 June 2019

Abstract

The results confirm the informal, dynamic and rudimentary software development practices shown in literature to be followed by startups. Requirements engineering is done ad-hoc, processes are tried out and either rejected or accepted and change as the context changes. This reflects the high degree of reactivity in startups. *Speed* vs *correctness* are competing qualities of the product development as the company goes from being an early stage startup to gaining maturity and stability. A step towards achieving *speed* in the *right direction* is to involve developers in product development through deliberate and systematic communication of information relevant to decision making of the product development.

Studies in requirements engineering in the context of agile software development and software startups analyze the field as a set of distinct, partly sequential, activities, typically elicitation, documentation, estimation, prioritization and validation. However the study reveals that the activities are done holistically, without strict frames. Such requirements engineering can be considered as being a process rather than a set of static activities: The relation between the field's constituents playing a larger role than its constituents considered in isolation.

Requirements engineering is an improper word for what it encompasses in the context of software engineering generally and software startup engineering specifically. I propose *Software Activity Handling* as the new term for the knowledge area encompassing requirements collection, elicitation, specification, documentation, prioritizing and validation.

Sammendrag

Forskningsprosjektet bekrefter de uformelle, dynamiske og rudimentære praksisene fulgt av oppstartbedrifter som finnes i litteraturen. Kravbehandling(requirements engineering) er gjort ad-hoc, prosesser prøves ut og beholdes eller forkastes ettersom bedriftens kontekst forandrer seg. Dette reflekterer den store graden av reaktivitet i oppstartsbedrifter. Fart kontra riktighet er kvaliteter i konflikt for produktutviklingen ettersom bedriften går fra å være tidlig-fase til å oppnå mer modenhet og stabilitet. Et steg mot å oppnå fart i riktig retning er å involvere utviklere i produktutviklingen gjennom bevisst og systematisk kommunikasjon av informasjon som er relevant for avgjørelser relatert til produktutvikling.

Litteraturen innen kravbehandling i kontekst av smidig programvareutvikling og oppstartsprogramvarestartups analyserer feltet som et sett av distinkte, delvis sekvensielle aktiviteter, som kravinnsamling, -utvelgelse, -spesifikasjon, -dokumentering, estimering, prioritering og validering. Dette studiet viser at kravbehandling er gjort helhetlig, uten strenge rammer mellom aktivitetene feltet består av. Kravbehandling som dette kan ses på som en prosess, snarere enn en samling statiske aktiviteter: Avhengighetene mellom feltets bestanddeler spiller en større rolle enn bestanddelene sett isolert.

Requirement engineering er en misvisende term for hva det innebærer innen programvareutvikling generelt og oppstartsprogramvareutvikling spesielt. Jeg foreslår Software Activity Handling som den nye fagtermen for kunnskapsområdet som består av kravinnsamling, -utvelgelse, -spesifikasjon, -dokumentering, estimering, prioritering og validering.

Preface

This research project has been done during the academic year of 2018/2019 as a cooperation between Norwegian University of Science and Technology and an anonymous startup called "Blueberry", a growing software startup building products based on machine learning. Hours of literature studies, observations, interviews, and analysis of these have resulted in the document you are now about to enjoy.

I would like to express my gratitude to the group of people allowing me to do this project - all the people behind "Blueberry". Witnessing the journey the company has gone through has been tremendously interesting, not to speak of inspiring. I wish you all the best, and good fortune in the years to come.

Next, my supervisor Eric Monteiro at the department of Computer Science at NTNU deserves my deepest thankfulness. As an experienced guide throughout the entire project period, he has provided invaluable tips and feedback on the path that has been created along the way.

Erlend Morthen June 2019

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

The startup scene keeps growing and so does the need for reliable and applicable knowledge. Research on the field is slowly picking up in speed and gaining maturity, but many questions remain unanswered for most startups, who strive to find purpose and direction in the chaotic world they operate in.

Whereas software startup engineering is a relatively new field of research, the knowledge area of agile software development just turned 18 years old. Although having different points of origin, they share some fundamental aspects, such as the search for reactivity and speed. The overlap between the two fields has grown further with the popularity of Lean Startup(Ries et al., 2012), calling for iterative working method.

Requirements engineering, a category of software development, a scientific field traditionally associated with extensive projects with large and detailed upfront requirement analysis phases, has recently become a subject of research also within startups(Startup-Organization, 2019). Previous research has referred to requirements engineering activities are amongst the most problematic encountered by startups(Klotins et al., 2019b). This is perhaps not surprising, as the road to determining what a startup should do goes through the field of requirements engineering - calling for further research in the field.

In addition to entrepreneurs and people with interest in software startups, this paper will hopefully be of value for any professional working with product development, managers, designers or developers.

1.2 Research Questions and Objectives

In order to frame the research project, one main research question:

RQ1: What are the practices of requirements engineering in a growing startup?

In order to treat this question, I will answer three sub questions:

RQ1.1: How does requirement elicitation and prioritization take place?

RQ1.2: How are the activities within requirements engineering related with each other?

RQ1.3: Are the activities within requirements engineering dependent on other aspects of software development?

1.3 Document Outline

The paper is organized as a traditional master thesis: Firstly, an introduction providing a quick impression of the research project, as well as a presentation of the research question. Thereafter, a review of the relevant literature is provided, followed by a chapter on research methodology. The data collected from the case study is then presented. The discussion, an analysis of the case results and the literature, is then provided, before finally concluding on the research project and suggesting further research in the field.

Chapter 2

Theory

2.1 History of Software Development Methods

As software development has gone from non existent to being an established scientific field during the last decennials, so has the need for structuring the software development. Software development occurred for the first time in the 1940s and 50s, mainly in fields of academia or military. Despite the nature of these organizations, methodologies as we know them today were practically non-existent; code was written ad-hoc, or principles for management were borrowed from other fields, with only a few exceptions(Benington, 1983).

A few decennials later, Boehm predicts that the software costs would exceed hardware costs in a large development project for the American air force(Boehm, 1983), and very large commercial IT projects occur for the first time. The need for optimizing software development(SD) becomes evident. Formalizing the planning phases before starting to implementing became was established as best-practice, as time spent on planning what to do would save significant time later, when doing it. The consensus tended towards a plan-driven approach.

The best-known example of such methodology is the waterfall model, originally presented by (Royce, 1987). Used commonly throughout the 80s and 90s, it is today generally looked upon as outdated. A common misconception sometimes leading to oversimplified and trivial conclusions of the type "hence is agile methods the better option", is that the waterfall model lacks feedback loops, implying a low degree of flexibility during the project cycle. Stated already in the original paper(Royce, 1987), the feedback loops were explicitly considered: The waterfall model, as originally described, allows feedback and reiterations between consecutive elements in the waterfall, as in figure 2.1. Water, interestingly enough never mentioned in the original paper, can indeed flow upstream in Royce's model. Despite the common

critic of imposing inflexibility to projects, the waterfall model is still in use with success(D. Linthicum, 2016), suggesting that some of the categorical critic found of the methodology in the popular literature has been undeserved.

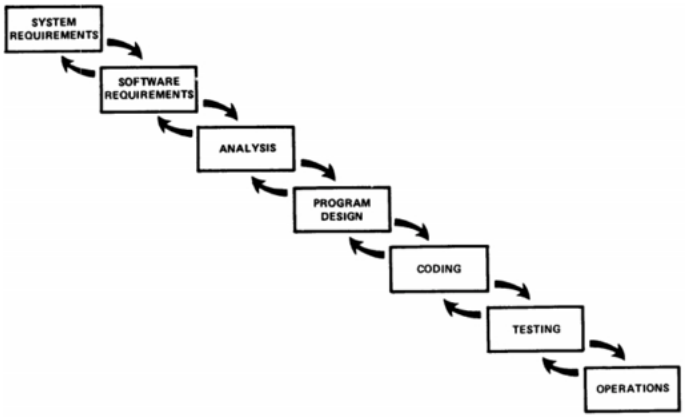

Figure 2.1: The Waterfall Model

Could there be too much of a good thing? Probably so, as one later could observe reactions to the somewhat rigid, plan-driven approaches. As the field of SD matured and competition rose between an increasing number of actors on the global development scene, new properties like speed and flexibility became new important properties for IT projects. One new methodology was the Spiral Model, presented in 1985, combining iterative work, i.e. doing a number of tasks in repeated cycles, and risk-driven development, in the meaning of letting the level of risk dictate what to do in the next cycle, and how. Each cycle, or 360 degree portion of the spiral, involves a number of defined tasks, such as determining objectives, alternatives and constraints, risk analysis and prototyping.

The work methodologies behind the large projects mentioned above implied a considerable work effort going to requirements and documentation. It also had the property of creating a sequential work flow, meaning finishing one part of the work flow before starting a new one. An oversimplified example of this would be to finish all planning of a software system before implementing it, and thereafter finishing all code writing directly contributing to the product before writing any tests. It can be tempting to ask the following questions: How does one handle changing requirements after the planning phase? When is the team going to adjust the product based on feedback, if any? Does the fact that one has not yet dived into the context through the implementation phase affect the quality and applicability of the documentation resulting of the planning phase?

2.2 Agile Software Development

The term "agile process" was coined in the early 90s by the research group "Agile Manufacturing Enterprise Forum, AMEF" at Lehigh University(*Creating the agile*

library : a management guide for librarians 1998), advocating for a transition from mass production to agile manufacturing(Nagel, 1992) in an attempt for USA to regain the industrial leadership lost during the previous decennies. Realizing the similarity to the Japanese *lean manufacturing* established in the 50s, they summarize the difference as "Lean is a response to competitive pressures with limited resources, agile is a response to complexity brought about by constant change." (Dove, 1993). Fairly clear, but the terms lean and agile even today get mixed up.

A few years later M. Aoyama used the term for the first time in the context of software development(Aoyama, 1998), but it was not until 2001 that the term got well-known and given a meaning that would resonate in the developer community.

As a reaction to the process-heavy approach of the 80s software development, new working methodologies aiming at improving the making of software for all stakeholders emerged. Extreme programming, SCRUM, Crystal and Adaptive Software Development where software development methods invented in the 90s having in common that they focus more on adapting to change and less on initial planning. The emergence of these methodologies should been seen in light of the era that software development(SD) was in at the time - the internet boom created demands for software at speeds the community had witnessed before.

This movement led amongst others to the meeting of a group of important developers and thought leaders in 2001 creating what got to be named "The Agile Manifesto"(*Agile Manifesto* 2001). The group represented a selection of modern methodologies. The agile manifesto's core values stated below are often referred to as a reference to agile software development(ASD) principles.

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

Sometimes mistaken for advocating the elimination of process overhead (the final clause of the manifesto "That is, while there is value in the items on the right, we value the items on the left more." being ignored), it expressed the signs of the new times in a manner that representatives from different new methodology schools could agree on, and has since experienced a tremendous resonance within software development community.

2.2.1 Scope

"When topics are fuzzy, underlining the boundaries and what lies within these boundaries become almost as important as understanding the intricacies of the individual constituents". The topic of ASD is indeed fuzzy: "To state that a particular

method is or is not agile is almost meaningless given the lack of consensus on what the term "agile" refers to" (Conboy, 2009). A recurring problem of identifying ASD is the lack of a clear-cut definition the software community agrees upon. The principles from the manifesto are frequently referred to, and it seems to be generally agreed that responsiveness, collaboration and a low degree of planning are central to ASD In this paper, the following definition(Abrahamsson et al., 2017) is suggested:

"What makes a development method an agile one? This is the case when software development is incremental (small software releases, with rapid cycles), cooperative (customer and developers working constantly together with close communication), straightforward (the method itself is easy to learn and to modify, well documented), and adaptive (able to make last moment changes)."

It might just as well be interesting to notice what is *not* part of the definition. Implementations of the above principles are not stated. This implies that standup or retrospective meetings, instructions for pair-programming, task management boards or other practical instructions are not part of agile in its essentials. Nor does it specify any degree of "process light or heaviness".

Concepts which are central to ASD existed in the software industry before the term agile was coined(*Agile Software Development: Current Research and Future Directions* 2010), such as the spiral model discussed above. Despite its popularity growing in the field of software development, agile methods have proven its relevance to other sectors(Sutherland, 2016). *DevOps*, aiming to unite development and operations, has experienced increasing popularity the last few years and arguably may have taken the buzzword-throne after agile. Significant research effort is done in the field of Large Scale Agile, as well as in company-wide agile adoption, not limited to software development.

Given the opacity of the theme, discussing agile in broad terms risks losing interest. On the other hand, investigating well-defined, specific aspects of SD, with ASD as a backdrop, makes sense.

Adopted to such an extent throughout the industry, agile methods have not lacked critics. One of the main points of critic is the evangelical approach some seem to have to ASD. The following sections present some of critics towards agile methods:

2.2.2 The Agile Hype

The massive adoption of agile development methods reported in the State of Agile Report 2017 can to some extent be explained by the hype that some claim the phenomenon has become. Organizations may fear being viewed as outdated and consequently label activities not having much to do with agile, as being agile. Other software organizations may state that they follow agile principles when implementing isolated sub concepts. Having a daily meeting while standing up, *standup meetings*,

does not make the software development agile.

Organizing software development, or organizing any other activity for that sake, is a field in which the optimal practices may not be evident. Involving social aspects, some may miss the clarity of natural sciences. Left in the "landscape of confusion", one may turn to commonly adopted methods in search of answers - the result being superficial, suboptimal adoptions of agile. The hype has led to criticism towards what some consider a dilution of the phenomenon(Tracz, 2015) - in the sense that if you can use a word to describe anything, as discussed in the "Scope of Agile" section, what is then the word's value?

2.2.3 Always Appropriate?

One of ASD's core advantages is the ability to adapt to change in either requirements or in the surrounding environment. Under the conditions where the requirements can be determined in advance, and of static requirements in a predictable environment, following a plan-driven approach will be advantageous, e.g. by creating more predictability in for the stakeholders(B. Boehm, 2002). The same goes for a certain type of environments - in (safety) critical projects, it needs to end with the fulfillment of a certain number of non-negotiable requirements. Under such conditions, the advantages of planning will outweigh those of adaptability(B. Boehm, 2002).

How software development best can be organized depends on the context, including the culture, both specific to the company and to what is specific to the country or region. ASD originally flourished in the United States, a country in many aspects different from where this case study is carried out, Norway.

One can argue that some of the needs creating room for the success that agile has been are stronger in the United States than in Norway. Firstly, agreements are more commonly based on formal contracts, in many cases leading to more overhead.

Secondly, the degree over hierarchy is lower in Scandinavian countries than in many others. A low degree of hierarchy promotes informal, and perhaps closer, dialogue, both internally and towards customers. This is further accentuated by the factor of organization or enterprise size. A small startup is generally less formal than an established corporation.

"Agile and traditional methods will have symbiotic relationship in which factors such as number of people in the team, application domain, criticality and innovativeness will determine the process to select."(Cohen et al. 2004), was stated in 2004, but one can assume that it still in 2019 holds its validity.

2.2.4 Chaos versus Agile

As agile was born as a reaction to overly process heavy methods, it happens that some justify the lack of planning by "being agile". This obviously is not correct - planning and structure are required for the core concepts of agile like customer cooperation and incrementality to be followed.

2.3 Software Startup Engineering

Software development(SD) in the context of startups has a number of fundamental differences from SD in established organization, affecting the choice of which development method to follow. Startups often live in a land of uncertainty, and the need for speed and adaptability are important. The popular "Lean Startup"(Ries et al., 2012), accentuating the value of iterative work and adaptability based on frequent customer feedback, hence shares some of the underlying principles of ASD: adaptability, or simply agility, and close customer collaboration, rather than upfront planning.

Current needs are prioritized over those of tomorrow to a greater extent than within large organizations. This gets visible through pragmatic technology choices, allowing for "hacky code" (Dalal, 2019) which provides working features today, but to some extent leaving technologic debt, i.e. issues which need to be dealt with later. Spending resources on writing code that allows for future scalability and modifiability may also be down-prioritized. While large organizations may have an "old" existing code base, i.e. legacy code, startups often start from at tabula rasa, providing more freedom in their technology choices. (Pantiuchina et al., 2017)

Software startups operate under conditions of extreme uncertainty, which is actually what defines a startup according to Ries(Ries et al., 2012). The world appears less predictable than in the environments of established firms. According to the principles of lean startup, this calls for incentives to deliver quicker and test new features faster than what one would in older companies. The *need for speed* calls for shorter iterations and frequent feedback, and less for thorough implementations which need to function properly forever. Taking into account that a number of the future features will prove to be irrelevant, extensive use of prototypes and beta versions makes sense.

Startups tend not to follow established process frameworks, leaving more liberty and allowing for customized and dynamic work methods in its ways of working.(Berg et al., 2018), (Klotins et al., 2019a). This is partly due to the changing environments under which they operate: The number and type of customer varies quickly, competitors may show up, the need for and number of employees with new competences can suddenly increase. Startups use lightweight process which allow for rapid changes in both its product and its process (Berg et al., 2018). Another reason can simply be a

matter of size - startups generally have few employees. Hence may frameworks for systematic information sharing and documentation lose its interest.

A recent multi case study investigating 88 startups, identifying the challenges amongst software startups presented the following challenges startups recurrently encounter: (Klotins et al., 2019a):

- A time-consuming and expensive mimimum viable product,MVP, is a sign of poor technology choices and over scoping, the first start-up antipattern.
- Lack of customer interest in the product could be rooted in a failure to establish a feedback loop earlier, the second start-up antipattern.
- Difficulties to scale product into new markets could stem from lack of organizational support, the third start-up antipattern.
- To save time and resources and greatly improve chances of a successful project launch in start-ups, the focus should be on better engineering, not more of it.

A slightly older, two-case study (Giardino, Wang, et al., 2017) of software startups which failed, presents the following reasons for doing so:

- The two case companies showed initial evidences of a lack of systematic feedback from customers to improve their market understanding.
- As team dimension is concerned, in order to have the focus and the ability to evaluate risks, Carmel suggests that entrepreneurs need to look for a well formed, skilled core development team, rather than just a set of product ideas and features.
- One of the key determinants of success in startup companies is the passionate behaviour of the founders.
- Indeed, over time learning progress slowed down and the two startups were only concerning how to gather more and more customers. However, improving customer acquisition before problem/solution fit has been premature, because users were not hooked to the product.

2.3.1 Phases of a Startup

There are a number of models for classifying startup maturity(Gralha et al., 2018). A framework proposed by Blank(Blank, 2003) suggests that companies first should make an effort finding the *problem/solution fit*, focusing on customers rather than on the product and, if applicable, its technology. This phase includes testing the riskiest

hypotheses by implementing a first solution. Only after iterations in this phase and gaining a thorough understanding of the problems to be solved, and how, should the startup move on to focus on building the product that will solve real customer's needs, i.e. finding the *product/market fit*. If the company does not achieve this, it should go back to phase one, or *pivot* (Giardino, Wang, et al., 2017).

A framework for classifying a startup's phase, still proving its relevance 20 years after being published, is the four phase model(Crowne, 2002): i) *Startup*, being the period between product conception and the first sale. ii) *Stabilization*, from the first customer takes delivery of the product, until when the product is stable enough to be commissioned for a new customer without causing any overhead on product development. iii) *Growth* follows and lasts until market size, share and growth rate have been established and all business processes necessary to support product development and sales are in place.

Known challenges to the growth period are that the product development depends on a small number of skilled individuals who become a bottleneck in all activities. Their skills and knowledge are not available in the market place. There is no plan to develop these skills in new recruits. Another challenge occurs when the product pipeline is empty, or when the company cannot meet demand for information on future product developments. Few product startups are able to transcend their first success and generate a stream of new products. "Whilst there is no substitute for inspiration here, this must be combined with a robust process to capture product ideas, develop them and choose between competing ideas"(Crowne, 2002).

iv) *Maturity* is, according to Crowne, the final phase in a startup life, and is characterized by the traits of a mature organization rather than a startup.

The hunter-gatherer cycle(Nguyen et al., 2015) is a different conceptual model of the evolution of software startups: "emphasis on a hunting cycle when generating ideas, eliciting requirements and customer development predominate, followed by a gathering cycle when the product-market fit becomes better understood and activities of requirements description, prototype implementation, automated testing and system integration dominate while the hunting continues, bringing innovation to the product".

2.4 Requirements Engineering

Before discussing the field of requirements engineering, it's worth looking at the name the field has been given, as the term is somewhat misleading: Requirements can intuitively be associated to the documentation of the expectations of a clearly defined artifact, rather than the broader meaning adopted in recent literature, such as "Requirements are a specification of what should be implemented in a software"(Heikkila et al., 2015). The second half of the term can also be seen as being

misleading: Despite being a field of knowledge arguably closer to process management than to technology, it, perhaps counter intuitively, is called an *engineering* domain.

2.4.1 Agile Requirements Engineering

Agile requirement engineering(ARE) can be defined as the following(Heikkila et al., 2015) "In agile RE, the requirements are elicited, analyzed and specified in an ongoing and close collaboration with a customer or customer representative in order to achieve high reactivity to changes in the requirements and in the environment. Continuous requirements re-evaluation is vital for the success of the solution system, and the close collaboration with the customer or customer representative is the essential method of requirements and system validation."

Two systematic reviews have investigated the effects of ARE: (Inayat et al., 2015) and (Heikkila et al., 2015). Both reviews expresses the recurrent advantages of ARE compared to traditional RE, as well as the new challenges introduced by ARE. The conclusions are generally similar, affirming the validity of the results. ARE generally solves the following challenges frequently encountered by traditional RE:

- Lowering the communication gap
- Overscoping: the RE requires more than the team can deliver
- Requirements validation: difficult to validate requirements before it's too late
- Unreliable and lengthy documentation: a massive document which is not updated
- Solving rare customer involvement
- A reduced tendency to overallocate development resources
- Responsiveness to change

, as well as finding that agile RE posed a number of new practical challenges:

- Minimal documentation and tacit requirements knowledge
- Customer availability
- Budget and schedule estimation, and imprecise effort estimation
- Inappropriate architecture
- Neglecting non-functional requirements

- Customer inability and agreement
- Contractual limitations and requirements volatility
- Requirements change and change evaluation

2.4.2 Requirements Engineering in Software Startups

Even though software startups not necessarily follow agile principles in their work flow, they have a common contextual denominator, leading to a few similar needs. Both startup and agile teams will have to decide what to implement in each iteration and which requirements will deliver the maximum value to customers(Melegati et al., 2019). Software startups are not agile by nature, but they share a set of common characteristics, and agile software development has become so popular and accepted that a considerable portion of startups will be inspired by what appeared radical not many years ago. Hence will literature from both startup engineering and agile software development be relevant for papers studying startup engineering.

A study investigating 88 experience reports from startups revealed, after coding the collected data and grouping after number of knowledge areas(KA) within software startup engineering, that *software requirements engineering* is both the most discussed, and having the most quotes related to negative impact on the product: Requirements engineering the major challenge, if not a pain point, for software startups(Klotins et al., 2019b).

(Gralha et al., 2018) provides a framework for categorizing requirement engineering in startups along 6 dimensions: Requirements artifacts, Knowledge management, Requirements-related roles, Planning, Technical debt and Product quality. Specifically for the dimension regarding *planning*, the report provides the three following phases i) In the earliest stage of a startup, planning in non-existent or minimal. Initial users/clients are found, and reactions from these early adopters is an important factor in the future development. ii) A growing customer mass leads to a richer backlog, and planning becomes monthly or quarterly. Schedules generally remain loose, releases often not planned. iii) The third and final phase is where planning becomes strategic and aligned with vision.

Requirements engineering has traditionally been associated with large, process-heavy projects, spending much time and resources upfront to the implementation phase. A list of final and relatively detailed requirements was provided in an early stage of the project.

The process of producing this is what is known as *requirements engineering*, and can be decomposed into a number of activities: *elicitation, negotiation, analysis, specification* and *validation of requirements*. Agile software development, addressing the problems associated with too much upfront planning through collaboration and

adaptability, should then be in opposition to the traditional take on the field. So how does requirements engineering play out in an agile context? Both agile and lean startup principles in practice lead to projects being executed in iterations, including deployment, testing, feedback and consequently possibly refinement or readjustments of the requirements. During the first iteration, only high level requirements are defined, whereas the detailed are only determined during the iterations to follow.

A 2019 report on software startup engineering presents the following critical remarks on startups practices: (Klotins et al., 2019b) "... However, the applied requirements engineering practices are often rudimentary and lack alignment with other KAs. As a consequence, inadequacies in requirements engineering hinder other engineering activities and might lead to unwanted technical debt, poor product quality, and wasted resources on building irrelevant features."

2.4.3 Market Driven Requirements Engineering

For many businesses, what is to be implemented is made for a known customer or user, or for a well-defined purpose. This can be the case for e.g. a software company writing a new internal application for a client company. The RE involved in this activity can be characterized as *customer-driven*(Regnell, 2005).

When the customer is not defined or known, for instance when being the first one to offer a product/service in a new market segment with uncertainty, or when offered to an open marketplace, the process of requirements engineering can be categorized as *market-driven*(Regnell, 2005). Software startups often find themselves in this category (Rafiq et al., 2017). One may differentiate between two main drivers for development of a product: *Market pull* i.e. a need from the market, and *tech push* i.e. new technology providing new opportunities previously unknown to the market. When the software produced is aimed at being offered to an open marketplace rather than to one specific customer, the RE process can be categorized as market-driven(Regnell, 2005).

2.4.4 Requirements Elicitation

Requirements elicitation represents practices to collect requirements and to identify sources from where engineers can collect requirements(*SWEBOK 2004*, 2005), also known as "requirements gathering". Technology focused companies often neglect user feedback in favor of inventing requirements internally. This is partly due to difficulties obtaining feedback on a new product that is unknown for a market(Medeiros et al., 2017), and partly due to focus on technology rather than actual customer needs(Karlsson et al., 2007).

Customer input is reported as an essential part of requirements engineering, even

though it is reported as complicated due to physical distance and vague understanding of the target market. A commonly reported difficulty is to create an engaged community of early customers of the product. This community facilitates requirements elicitation, validation and other activities were direct customer feedback is essential.

Collecting requirements from defined, explicit clients may prove to be easier than in the market-driven context. "Traditional requirements elicitation techniques, in which elicitation sources are customers or users, are not appropriate to elicit market-driven requirements, because in market-driven requirements there are no specific customers or users until or unless the product is released or at least beta version is handed over to a set of customers. Often market-driven requirements are invented initially. The invention process is inspired by business strategies and the vision of a product-to-be."(Klotins et al., 2019b)

Some reports have stated concern in that agile software development lacks concern on data structures: "There is concern in defining the conceptual model in a systematic way together with user stories. Sometimes, the data entities are generated from the classes defined in the source code. It may end up creating an unstable data model"(Inayat et al., 2015).

2.4.5 Requirements Documentation

The hundred long pages requirements documents resulting from extensive upfront analysis pages are less common nowadays. Jørgensen(Jorgensen, 2016) concludes that a recurring factor in successful large IT projects is flexibility and agility, including extensive documentation.

Startups and agile project share characteristics when it comes to requirements documentation. Startups tend to have shallow documentation, and so does agile teams, compared to traditional approaches(Melegati et al., 2019).

A common tool for requirements documentation in agile software development is user stories, expressing wishes on the format as a X I want to see Y, such as as a user I would like to see the weather forecast, or as an administrator I would like to edit member's permissions. Widely accepted as format for documenting requirements as of 2019, it provides enough information to developers to implement software. It's worth noting that use cases state what is to be developed, not how: A use case will never specify the technical architecture, and most will consequently be insufficient as documentation in most cases. User stories are targeted to customers and cover simple functional requirements. They do not address system and non-functional requirements that are also required for coding, testing and maintaining(Medeiros et al., 2017).

2.4.6 Requirements Prioritization

Release planning is related to requirements engineering, and requirements prioritization in particular(Klotins et al., 2019b), as it concerns the set of requirements to be delivered to the customer and provide competitiveness to the market. Klotins distinguishes between two main approaches startups follow when planning product releases: 1) frequently releasing small increments and 2) delivery of a fully-fledged product. The former is related to continuous requirements validation and may allow for quick product direction adjustment. Challenges for following this approach are the overhead associated with each release, and dependencies between them.

The latter involves a risk of having put the effort of implementing irrelevant features. Companies may want to polish their product before release, wishing to keep or attract customers in the competitive market it likely operates in. Companies collecting early user feedback and have launched a less complete product, report fewer difficulties in marketing the product(Klotins et al., 2019b).

More frequent releases has the advantage of allowing for more frequent feedback on features, if these are collected and measured. It is hence related to requirements validation, i.e. does the client or user accept what is implemented. Frequent releases often implies less risk associated with each delivery - each release containing fewer changes. A disadvantage is the overhead associated with each new delivery, including the risk of errors related to the release.

A recurrent method used by startups is to have a higher level management define *milestones*, then let development or product team determine the task implementation orderMelegati et al. (2019).

2.4.7 Non-Functional Requirements

Requirements may generally be categorized in functional and non-functional requirements(NFRs), sometimes called quality attributes. Whereas the requirements in the first category are reflected in new functionality to the product, the latter concerns aspects such as usability, security, performance or compatibility.

Neglected non-functional requirements is found to be one of the challenges of agile requirements engineering(Inayat et al., 2015), partly due to the extensively used user stories, generally covering only functional requirements.

Chapter 3

Research Method

3.1 Data Collection

3.1.1 Research Phases

The data collection period lasted throughout most of the pilot study, from September to December 2018, as well as during the master thesis, from January to April 2019.

The pilot study was to a large extent spent acquiring a foundational understanding of the case as well as the theory. Scoping the research project proved to be amongst the major challenges, and the main themes and research questions were determined towards the end of the project.

The master thesis was a continuation of the work done during the pilot study. Understanding of both case and theory was brought into the thesis. The themes determined during the autumn, i.e. documentation on estimation and requirements, as well as agile software development in established organizations, served as a starting point for the new phase.

3.1.2 Principles for Data Collection

Yin(YIN, 2011) provides three principles to follow in order to improve case study research quality:

Multiple Sources of Evidence

Data triangulation is the rationale for using multiple sources of evidence: Any case study finding is likely to be more convincing if it is based on several different sources. Information sourcing from one document or interview is not sufficient to base an information on. This was achieved by backing up claims from interviews with relevant data from documents or by observing discussions of same themes during meetings. Most often was done by collecting various' peoples views on the theme during interviews, in search of patterns or to confirm that there was no pattern. An examples of this is the question on how the roadmap is interpreted by different team members: open ended questions about the roadmap was asked during different interviews, the document itself was investigated, and roadmap related meetings were observed. In sum, this lead to a solid base for understanding and analyzing the subject.

A Case Study Database

Case studies have been criticized for lacking transparency regarding what and how raw data led to the themes and conclusion. One counter measure of this is to separate clearly between the processed data in the report, and the raw data, as well as making the raw data available for the reader. All interview records, meetings notes, and a log of other interesting events throughout the data collection period are available at request, but for confidentially cannot be published.

A Chain of Evidence

The collected material is documented in the *Data Collection* chapter and the most relevant parts of this is repeated in the *Discussion* chapter.

3.1.3 Data Sources

Six sources of evidence for case studies are presented in Case Study Research (YIN, 2011), as listed below.

Archival records were not relevant to the project as the relatively young company had few or no archival records available. Physical artifacts are objects such as devices, tools or an instrument, and were considered irrelevant to the research of a digital company.

The below sections describe the used data sources:

Data Source	Used
Documentation	*x*
Archival records	
Interviews	*x*
Direct observations	*x*
Participant observation	*x*
Physical Artifacts	

Figure 3.1: Data Sources

3.1.4 Documentation

A number of documents served as sources of data for the research project. Calendars were used to investigate if, when and between whom meetings would take place. The task management system Trello provided data in a number of ways: Feature documentation, task estimation, how features were split into tasks and how task relevant information was communicated through the system. The documentation platform Google Drive was used to look into design documents, as well as some architectural and technical documentation. The communication channel Slack, was rich on information and provided insight in discussions on features, and equally important, the culture of the company. Finally the roadmap viewer on the web platform Airfocus documented the roadmap.

3.1.5 Direct Observation

Direct observation is done by a passive observer. During the research project it took place as meeting observations which were held throughout the project period. A weekly tech team meeting, in which team wide themes were discussed, was the most currently observed meeting. Other observed meetings included gatherings of developers working together on a feature and team(a vague concept) meetings.

Whereas the interviews described below always would lead to generation of interesting data, it could happen that meeting proved to have little relevance to the research project. The observer being a fly on the wall, the themes discussed were not influenced by the researcher. 6 meetings where (directly) observed during the pilot study, 9 during the master thesis.

3.1.6 Participation Observation

Being employed as a part time employee during the most of the project led to both advantages and challenges for the research project. Distinguishing between the role of an employee and that of a researcher was explicitly expressed towards colleagues early autumn 2018. Time spent working was clearly separated from time spent researching - for colleagues to know when they had to deal with the different roles. The agreement of the two disjoint roles proved to be theoretical: In practice, knowledge from work time inevitably, and luckily, affected the research. Meetings being attended as an employee typically would generate ideas for further research.

Getting familiarized with the company and other employees was a major advantage for the data collection. Blending in during meeting observations was easier, and a tone of trust could be established during interviews. The issues of participation are discussed in below in "Reflections and Research Limitations".

3.1.7 Interviews

Interviews were the main source of data. Before each interview, an information letter, approved by NSD(*Norsk senter for forskningsdata* n.d.), about the research project was handed out, stating that the participation in the research project is voluntary and anonymous. Questions were intended to be reaching wide, allowing for unexpected answers or reflections. Interviews were typically ended with "any other comments concerning what we have discussed?", to which interesting statements often followed.

Perhaps did interviewees see the interviews and the research project as a medium to communicate opinions on organizational issues, which would be rarely discussed during the daily work.

A total of 16 interviews were held throughout the pilot study and master thesis. Interviews have been the main source of data for the research project. Several interviews were held at the company office, lasting from 20 to 90 minutes. Interviews were semi structured: Each interviewee was encouraged to "freely speak his mind".

The interviewing skills improved significantly during the data collection period. A main factor was the recording and transcription of interviews during the last months of the project. Recording the interviews rather than writing quotes down improved attention, and better questions. Transcripts led to more precise registration of the statements. Data which at interview time would not appear interesting might do so later, retrievable in the transcripts.

Another factor was the increased understanding of the subjects being discussed. Theoretical and general knowledge improved, and so did the understanding of the

case being studied. This was reflected in better questions and understanding of the answers during interviews.

Tables 3.2 and 3.3 summarize interviews and interviewees. Interviewees are identified with same name in the pilot and the thesis, e.g. alpha remains the same throughout the project.

Interviewees, Pilot study	
Alpha (two interviews)	
Beta	
Gamma	
Delta	
Total Interviews	5

Figure 3.2: Interviewees during pilot study

Interviewees, Master Thesis
First iteration
Alpha
Gamma
Epsilon
Zeta
Theta
Iota
Second iteration
Alpha
Gamma
Epsilon
Kappa
Mu

Figure 3.3: Interviewees during master thesis

Some individuals were interviewed several times: alpha(four times), gamma(three times), epsilon(twice). Repeated interviews typically went further in depth of themes discussed earlier, or on themes containing conflicting views between colleagues.

All interviews were semi structured, but they were gradually focused on narrower themes with time throughout the project, going in depth rather than width.

3.2 Data Analysis

Qualitative data analysis is the determination of patterns and themes, or categories, within the data. The categories can be determined based inductively on the collected data or based deductively on existing theories found in existing research. Most often, as is the case of this research paper, are the categories determined by a combination of the two.

The determination of the categories was a process spanning from the beginning until a few weeks after the data collection period:

Data is not collected completely randomly. During meetings, a researcher notes what he/she considers relevant for the research project. Among the vast amounts of documents available, only a few are chosen as basis for the research. And interviewees respond to the questions being asked by the interviewer. Thus will the data collected inevitably be coloured by the researcher already at "the source".

A typical scenario of this is that a quote occurring at a meeting or interview triggers the researcher's curiosity, and lead to related questions during the same or later interviews, generating more data about that theme, i.e. inductive data analysis. A theme may also generate research on related existing research, i.e. deductive research, which in its turn may turn into new questions for interviews. Thus are the categories a result of the interaction between the inductive and deductive approach to data analysis.

The categories have varied throughout the research period spanning from September 2018 to June 2019, and so has the understanding of the theory behind it. Finding a correct scope for the research project unfortunately was a challenge during the a surprisingly large portion of the research period. This was party due to a lacking understanding of the field in the beginning, and the preparatory research project largely consisted of getting insight in both the theory of the field, as well as getting necessary insight in the case.

The scope initially started as agile software development in a growing startup, but it would show that the theme was too broad, combined with the fact that agile software development is a vague theme, both in the current research and as perceived by the research subjects. A number of categories, related to agile software development(ASD) in various degrees, were determined and discussed. The categories, and the theoretical insight who followed served as a platform for further research during the master thesis period lasting from January to June 2019. The research themes for the thesis were yet to be determined. Inspired by what agile principles tell us about work flow, an iterative approach was chosen to the master period: two iterations of a "data collection - literature research" cycle was done. After the first cycle, a number of categories were determined, partly overlapping with the ones from found in the pilot study.

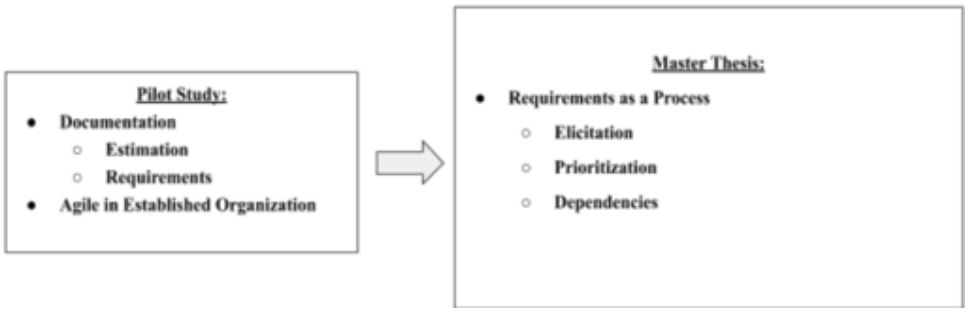

Figure 3.4: Research Themes

Literature research led to insight in the decomposition of the requirements process into defined phases: elicitation, prioritization, estimation, documentation, amongst others. Together with the statements and occasional disagreements of the research objects, a main research theme of *requirements as a process* was selected. The data indicated exactly this: Requirements undergo phases in which they are handled in different manners, and debated upon. One requirement could be interpreted differently by different members of the team.

An important characteristic of the theme is that it assumes requirements to be a process, i.e. dynamic, rather than a static product. For an fruitful analysis to take place, dependencies between the phases, as well as with activities outside the boarder of requirements engineering must also be taken into account. The term *requirements* is here used in a broad sense, including wishes of all formats from clients, ideas for new features to the product from within the team, non-functional requirements and even software bugs.

After the first iteration of the two iterations of the research work in the master thesis, one category was chosen to serve as main theme and perspective for further data collection and literature research: *Requirements as a process*. Data from what was originally considered another theme, as well as from the pilot study, such as Estimation, would also become part of the main theme, by being relevant to the main theme. Seeing this data in the light of the new main theme, as well as with new insight in related theory, renewed its relevance to the research.

3.3 Reflections and Research Limitations

One method for evaluating the research project is by following the seven principles described in (Klein et al., 1999) for evaluating interpretive field studies in information systems, serving as a framework for assessing the case study:

The Fundamental Principle of the Hermeneutic Circle: Hermeneutics is the branch of knowledge related to interpretation. The hermeneutic circle, foundational to all work of hermeneutic nature, suggests that our comprehension constantly moves

from seeing "the thing" as a whole, and understanding "the thing" by considering its parts. Throughout the study period, the researcher's knowledge on ASD, requirements engineering and software engineering has been changed and improved. Consequently, so has the understanding of both the literature and the case under study. The understanding of the subject as a whole, has in its turn influenced the understanding of elements of the research project. Several times, the same texts or quotes have been interpreted with new eyes, reflecting new understanding of the subject.

There were three main phases of improved understanding of the research topics: the pilot project and the two iterations during the master thesis.

The Principle of Contextualization Suggests that research needs to include critical reflection on the background of the case, so that the intended audience may see how the current situation emerged. Highly relevant to this case, information on the background of the case has been included in the "Context" section of the "Case Results". A challenge of the research project has been to separate between data (a) irrelevant context, (b) relevant context, and (c) having direct relevance to the research theme.

The Principle of Interaction between the Researchers and the Subjects This principle requires critical assessment of how the results were socially constructed through the interaction between researcher and subjects. The researcher's own ideas must be critically assessed for research to be valid.

A deliberate effort was made by the researcher during interviews in order not to ask questions which would generate answers being advantageous for the research project. This was partly achieved by asking open-ended questions, in the shape of objectively presenting a defined theme, and asking for thoughts or opinions related to the theme. Further in depth questions would be asked depending on what the interview subject would state.

Throughout the majority of the project period, the researcher has been spent the days at the office. The advantage of getting unique insights in the case is contrasted by the inconvenience of losing objectivity while studying the case. A factor further accentuating this inconvenience is the fact that the researcher has been employed as a part time developer in the enterprise being studied during the research period. A significant and deliberate effort has gone into separating these roles. It has been clearly communicated, amongst others by separating between "research days" and "work days".

Principle of Abstraction and Generalization The principle is especially interesting for case studies with few unique, or isolated, properties like the one described in this paper. From the concrete observations and consequent analysis, what can be induced as lessons for the general case, providing new insights for future cases? The general perspective is presented both in the intro and in the conclusion.

Principle of Dialogic Reasoning The impression and understanding of require-

ments engineering and agile software development evolved during the research period - from superficial and without having much experience to a thorough insight in the field. This was due to the data collection phases, the investigation of relevant research, and the play between these, as explained in both "The Fundamental Principle of the Hermeneutic Circle" and in "Data Analysis".

Principle of Multiple Interpretations In interpretive research, one allows for multiple subjective realities, being highly relevant for a case study like this one. Research subjects may for different reasons have different perceptions of one thing. It is the researches work to allow for and interpret the different versions given. This is especially relevant for the historical reconstruction of this study, mainly based on interviews.

Principle of Suspicion There may be systematic bias in the data collection. Tech team members of Blueberry may for example present versions of phenomena providing prettier images of themselves than what others would do. The role of the product manager and tech leads were especially prone to this - this research paper to some extent being an study of their work.

Chapter 4

Results

This chapter reflects all the data collected during the case study which is relevant to the research. First comes a section providing information on the context of the company, including a short summary of its history. Next comes the section on requirements process: starting with placing the relevant activities in context with each other, as well as in the perspective of other aspects of software development, before looking at separate aspects of requirements engineering: Elicitation, Documentation, Estimation, Prioritization and Validation. Data collected concerning non-functional requirements, an own category of requirements, is treated separately.

4.1 Context

4.1.1 Historical Background

The anonymous software startup "Blueberry" was founded in late 2016 by two individuals who currently have the positions of chief exectutive officer CEO and chief product officer CPO, and quickly experienced growth. The company offers products based on machine learning technology currently made use of by a large number of customers. The main field of application is in customer services - automating part of the service requests by end users, it frees companies' resources previously spent manually responding to users' questions. Doubling its number of employees during the summer of 2018, it currently employs approximately 35 people, of which 15 are tech team members, i.e. developers, designers, product manager or CPO. Other employees include sales, content writers and other managers.

The total developer workload is to some extent a reflection of the number and size of new clients. Adapting and integrating the core product with new clients' platforms require work, as well as preparing the platform for more traffic. Some developer

workload is independent of the number of clients - new features need to be added to the platform, as well as improvements of the machine learning algorithm, the core technology.

Blueberry has gone from being an early stage startup to being a mature startup. The rest of this chapter contains the results from the data collection, with a description of today's practice as well as of Blueberry's past.

4.1.2 Evolution of Blueberry's Tech Team

Blueberry's software development can grossly be divided into three main periods: *the early startup period, the establishing period* and *the mature startup period.*

The *early startup period* lasted roughly from the startup's very beginning until late 2017. The tech team was generally informally organized, with the number of tech team growing from 1 to 5 people.

"Things were less organized than today, everyone worked with a bit with everything. Meetings and discussions were done when needed, across the desk", states one of the early developers, and continues "The workload was generally reasonable, but could exceptionally grow bigger with very long evenings. A few times dangerous bugs did occur, without this causing too much trouble".

Documentation was written when needed for external communication, but rarely and not systematically for communication amongst colleagues. Tasks where informally defined, estimated and prioritized.

The *establishing period* lasted approximately from late 2017 until the summer 2018, marked by the engagement of new and previously experienced tech members. The company consequently experienced important changes, the new team members bringing new competence to the team.

"Deployments were too risky and tests rarely implemented. Errors which could have been avoided occurred. The early developers were competent, but generally lacked experience with handling larger software systems" states one of the members engaged during this period, and adds that the risky deployments were suboptimal as it lowered the delivery frequency and hence slowed the development. Consequently, a deployment pipeline with automated checks and tests was set up, formalizing mechanisms which earlier had been done inconsistently. Tests got more common, and separated testing environments were created. Requirements to documentation on each git pull requests were introduced, standardizing the development further. "Urgent fixes, putting out fires, subsequently got rarer than prior to the introduction of the new employees" confirms an early hired tech team member.

Except from git pull requests documentation, the tech team generally had little

formalized documentation. Estimation was not formalized and requirements handled ad-hoc. The same was the case for sprint organization, or more correctly the absence of such: tasks were generally effectuated when needed and at clients' deadlines, without setting internal temporal limitations for work units.

Together with a growing number of clients, an external factor affecting the workflow was the startup's funding mid 2017, when Blueberry got additional financial support from an investor. From being dependant on clients, implementing features at their request without much negotiating power, Blueberry after the funding had a financial margin to rank features in a strategic manner. From basing projects on clients' request, living from hand to mouth, Blueberry could plan strategically for themselves, resulting in new organizational challenges.

Some characteristics have marked the company since the beginning. It's worth noting that the tech team is culturally homogeneous. Males, sharing a relatively similar cultural background, they have chosen to work in a startup. The cultural homogeneity of Blueberry affects the office culture and consequently the ways of communication. A shared cultural background facilitates informal communication and suggests that the need for formal documentation decreases.

"For now it's comfortable, but we may miss out of acid tests down the road. The shared informal knowledge of heterogeneous, often small organizations is highly efficient, but will probably be inconvenient during future knowledge transfer to new employees or external agents", states the product manager.

4.1.3 Tech Team Members and Size

Attracting clients and investors, the startup has grown and undergone fundamental changes, and still does. While the tech team, i.e. the developers and designers, earlier consisted of a few early hires, it currently has employees in a conventional sense, like considering the work as a job rather than a life project. This includes working predictable hours and having a predictable salary.

A majority of the tech team members are employed directly from universities, which bachelors degree, master's degree or PhD: Out of 15 tech team members 6 have significant previous work experience. Except for the use of one technologist during a period mid 2017, Blueberry has not employed any external resources.

In August 2018, Blueberry hired a product manager(PM), responsible amongst other things for coordinating the increased development workload and product related client contact.

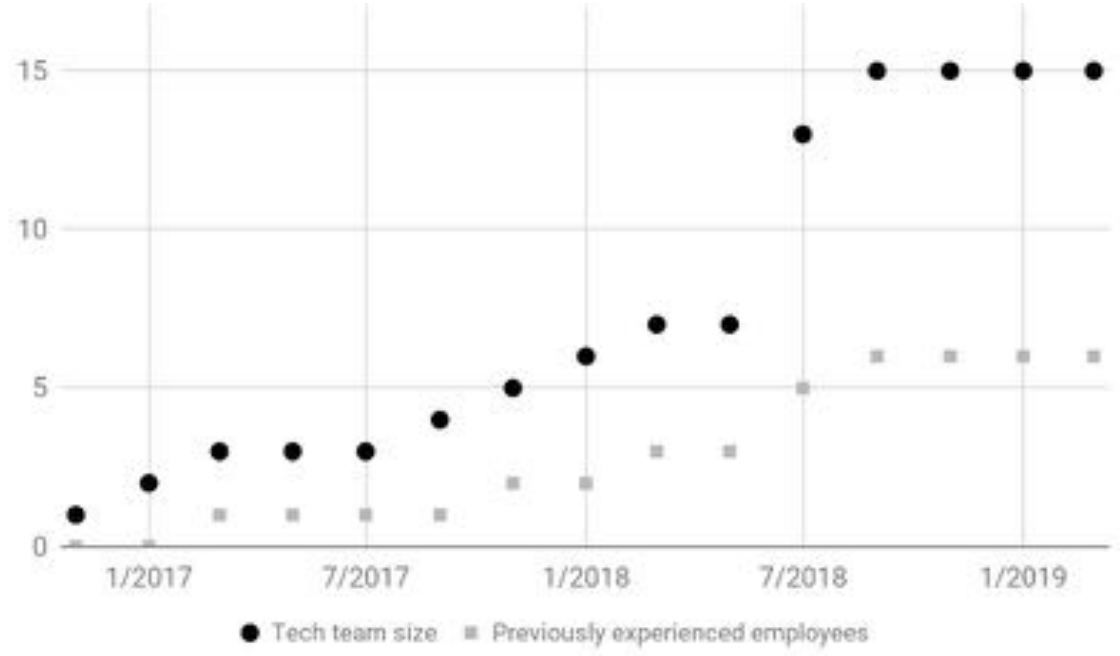

Figure 4.1: Evolution of Tech Team Members

4.1.4 Product

Due to the anonymity of the company, specifics on the company and the product cannot be stated here. With the exception of a few other activities, the tech team's activity consists of working on and improving the product platform. It is on this platform the content is added, of which the end users get value from. The content are generally specific to each client, and one client may use a number types of content. Both internal and external, i.e. employed in Blueberry or nor, content writers work on the platform. Internal content writers work only a few meters away in the open landscape office, allowing for quick communication and easy testing of potential changes to the platform made by the tech team. When a type of content is ready to be launched the client releases it as part of its web site, and the end user uses the integration with Blueberry's product on the clint website.

The product is thus a combination of the platform and the content and abilities of the types of content, added by Blueberry or by the client. Early April 2019, a CEO states both towards customers and in internal meeting that "the end user is to be in focus" for the next period, indicating that the company is keen on achieving constructive interaction between both customers and end-users. The ones paying for their service are other companies, thus making it a business to business, B2B, agent.

Blueberry is a *product company*, the platform being its main asset. This affects the relationship with the clients. As will be described later may clients' wishes for the product differ from what Blueberry believes to be the best in the long run.

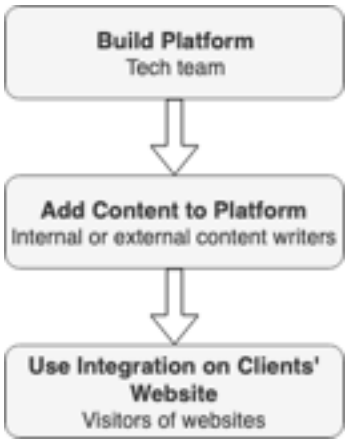

Figure 4.2: Product Creation Flow and Related Agents

4.1.5 Personnel Structure and Workflow

"We have tried out different ways of working since Blueberry was launched, but we're still without a formal process. We need to adapt to individuals, who differ in how they work optimally. There's much good intuition in the team, that we rightly should make use of." "Neither do we have someone, or a lead, that knows a given methodology really well. If there was such a person, forcing such a process over our heads, it could work out really well for some people here, and not at all for others". He continues "We're flexible in that we adapt, both product and process, to new situations."

The same message was expressed by a significant number of tech team members during interviews - Blueberry has been, and still is, flexible to its approach to structure and process. "We try something out, and if it doesn't work out, we adjust quickly", states another tech team member. This was confirmed through observation: Work methods were flexible rather than written in stone.

"I suspect we are more difficult to manage than the other devs", states one of the two developers in the team responsible for developing a core technology for the product. "What we're doing is basically experimentation, and it's difficult to know what a new idea results in, not to speak of estimation." He continues "I'm not quite sure who's actually in charge of us now."

Two tech leads were chosen in january 2018 among the tech team members, to do coordination work. "Having tech leads allows for quicker decisions, we don't have to discuss things for weeks anymore", states one of the tech leads.

The product manager that was hired from outside the organization during the summer 2018 was intended to do much of the tech leads' tasks. For a number of reasons, this did not work out as intended: "The PM's role was too largely defined, and he did not have time to follow up on the tech side. Also coming from outside the organization,

jumping in as the head of a team probable added to some disagreement. He tended towards a top-down style, while the tech team generally wished it otherwise.“

4.2 Requirements Process

Through data collection as well as through insight gained through investigating literature on the subject, it was found that the data could be collected and analyzed through the lense of Requirements as a Process. This implies considering requirements, defined below, as dynamic rather than being a static or final product. In Blueberry requirements were indeed dynamic and flexible: Modified as they went from an initial idea or request from a customer until finally implemented by one or more developers.

4.2.1 Definitions

A *feature* is a functionality on the platform.

A *task* is a work unit to be done by a tech team member, generally represented by a card on the task management system Trello.

A *requirement* is, at its most basic, a property that must be exhibited by something in order to solve some problem in the real world. It may aim to automate part of a task for someone to support the business processes of an organization, to correct shortcomings of existing software, or to control a device, amongst other things(*SWEBOK 2004* 2005).

4.2.2 Agile and Requirements

Before going in depth into the stages of requirements engineering, it's worth taking a look at what the research objects thinks of agile software development(ASD): Tech team members' view on what agile is, and whether Blueberry follows agile principles or not, varies and reflects that the term agile has a vague meaning. “What is agile?”, one person asks, while another states “I hardly dare to use the word agile, it can mean anything”. A third person states “Modern tech startups to some extent implicitly follow agile principles - also inspired by literature such as Lean Startup”(Ries et al., 2012).

However, one of the disagreements within the company, on the degree of up front planning and specification vs iterations and flexibility, goes to the core of ASD - upfront planning and documentation vs iterations and flexibility on the other hand.

“Agile is nice and reactive, and one gets to develop lots of nice and user friendly

features. But how do these features relate to the direction one wants the product to go in? I don't think it takes into account all the context necessary for the right decisions to be made", and continues "agile is nice in the short run, but lacks the view of the long run big picture", states the product manager.

The PM expresses concern with the prioritization of requirements when implementing in an iterative and feedback based manner: "what the users want is not equal to what the long term goal is. Developer don't necessarily have the holistic approach to product development".

Speaking more generally on agile, he expresses "Following agile for the sake of having a well determined development method is not interesting. Whats works the best in practice is interesting, not the academic purity of it. Product output is all that matters, in the end".

"As I see it, iterative work and MVP bases workflow is the manifestation of agile", states a tech team member. The following section treats exactly this, documenting how minimal viable product(MVP), iterative work and upfront planning are dealt with in the case.

4.2.3 Minimum Viable Product, Iterations and Upfront Planning

"The pm had a different approach to iterative work vs upfront planning from many of the tech team members." states one tech team member. "Time was spent in management planning upfront and detailing requirements, time that could have been used implementing the stuff. The detailed requirements did partly prove to be unuseful, as they further down the road could prove irrelevant.", he continues.

"We generally have relatively few really hard deadlines, compared to other companies.", states a tech team members, perhaps indicating that he has a different perspective than the product manager on relation. "I believe that high speed solves many issues, through shorter feedback loops", he continues.

At three occasions during meetings within the tech team, tech team members with special responsibilities pushed ons keeping ambitions for the first versions of new features down. "Let's make an ugly first version, and see how it works out with the internal users.", one member expresses during a meeting. "We're very inspired by the lean-startup approach", another tech team member states.

"I'm very fan of the MVP way of working with features, it makes develop what's need needed, not more. You reach the goal quicker than if you spend time on planning upfront."

"For certain tasks, it's not easy to work iteratively. For certain core product features,

the entire feature needs to be implemented before it can be tested - large amounts of data is needed, the infrastructure needs to be in place, etc.", states a developer.

"Also when clients are very clear and specific in their requirements, we try to implement features iteratively. The wishes are seen as part of the bigger picture - perhaps the need of the client can be met by solving the problems by other means than what they expected, means which also solve other customers' problems. Thus, it makes sense to work iteratively on these, originally clear cut and static, tasks as well. For new features that are not related to the product and anyhow would not affect other clients, such as integrations, I guess it could make sense, but these tasks are rare".

"What is an MVP, really? I think we lack some reflection on this, and a process of defining this. What the MVP varies from feature to feature, and it's difficult to define. It's true that the scope of the MVPs has swelled at several occasions", states a tech team member.

An example of this was occurred during the autumn 2018: A new feature was to be released, the final deadline being a presentation for partners and clients of Blueberry, this presentation being a few months later. The feature was initially a renewal of the interface design, roughly estimated to require a month of work. The scope of the delivery increased in scope throughout the project. "The project lived in its own universe. There were no releases during the project."

Another developer states that as that feature included an upgrade of the underlying framework, making it difficult to put into production only parts of it.

"Productivity was low at the end of the project, devs were working on tasks with low priority, perfectioning a feature that had not yet been launched", states a tech team member, and continues "We are aiming to release more often to customers now. We should get better at iterating over features - together. Alone, one typically loses perspective", states a tech team member.

Both MVPs and entire features have at a number of occasions become more extensive than anticipated. "It's unfortunate, as it slows us down" states a tech team member. "A consequence of that requirements aren't clearly stated", states the PM.

A member of the core technology team states that they generally have a list of features and improvements they work on. Another member of the team explains it differently: "There's no formal process on how we handle our tasks in the core technology team (of two developers). It's mainly just experimentation", states one developer.

4.2.4 Requirements Elicitation

Product Roadmap

The roadmaps is a overview of the new main features to be implemented during a given time period in the future, normally from three to six months ahead. It provides some five to fifteen items to be worked on simultaneously.

The roadmap is determined by the CPO together with a product manager, tech, design and sales leads. Elements to be added in the roadmap is based on the following sources:

- Analysis of competitors
- Feedback and wishes from clients
- The vision - "what culture do we want to be"
- Customer support channels
- Ideas generated internally

These sources overlap with those of the trello cards. Roadmap elements are debated between "managers and leads" before being determined, whereas some of the trello tasks are less refined.

"We've used the roadmap actively since the end of 2018, and it will hopefully become a (even) more important part of the developer's consciousness", states a tech team member.

In addition to roadmap elements, there are a number of future features which have been discussed and planned for. These elements are registered in the same platform as the roadmap.

Product Vision

Several tech team members expresses that the company's own wishes for the product, compared to those of the clients, are selected and prioritized higher than before. "In the early days, there were no clear product vision. Currently, we have the resources, both finances and resources, to choose for ourselves what the next feature should be. ", states the CPO. "We now have the guts to challenge or customers and even turn down requests for features, and I'd say that's a major strength", states the PM.

An effort is made in order to determine and communicate the future of the product on a higher, more abstract level. The company has meetings where all people involved

in the company, including investors participate communicate and discuss the future of the product. Still, as of late March 2019, the product vision statement was still in the progress of being made and communicated within the company. There is come disagreement on the degree of communication of the future plans for the product "Our roadmap spans a few months ahead of time. I can't say we have vision other than that", states a member of the sales team.

Other Requirements

While the main features are elicited and prioritized through the roadmap, many of the tasks do not originate from the roadmap. These tasks are

- Bug fixes
- Wishes for smaller features from inside or outside the company
- Requirements from clients which cannot wait until the next 3 month slot in the roadmap is ready
- Tasks related to non-functional requirements

After being handled, if needed, by one or more tech team members, tasks are consequently put in the "Incoming" column in a Trello Board. "Passing new tasks through the Incoming column gives us some slack and allows us to lower our shoulders while working. It makes us not forget things and it's a quite easy method to prioritize items. Seeing the tasks from a distance, and discussing them with others might put them in a different light. Perhaps the task wasn't as important/urgent as one first thought?", states the PM.

One the other hand, this process adds a delay to the implementation of a feature: the time until the next weekly prioritization meeting, and potentially the time until a developer is available for a new task.

The PM expresses some concern when requirements are communicated directly from clients or sales to the developers. "We try to avoid sending requests directly to the tech team, it should rather be processed by someone who has the overview. It's ease to get stressed up a client contacts you with requests or issues. One may forget to ask questions such as "How important is this, actually? How important is this client, really?"

Some work that is done can be of considerable size, but still neither elicited by the team nor formally documented or prioritized. "What I work on now is on my own initiative, I'm now working on it together with a colleague. That kind of work is typically done in the evenings or weekends, so that I can prove that it's working" The task was related to the changing the underlying framework of a part of the platform, making future features or changes easier to implement.

4.2.5 Requirements Documentation

Feature requirements were during the autumn 2018 documented on a trello card representing the feature. Such a Trello card would serve as base for new cards representing subtasks of that feature, each subcard being linked to the main card, for traceability and task logging. The requirement was structured in three sections *What*, *Why* and *How*. This system was introduced by the product manager, and implemented for most features and teams, but not all. After the beginning of 2019, when tech leads were appointed and the product manager moved into a role closer to client contact, this system ceased to be followed - reflecting the different views on the topic in Blueberry.

The roadmap mentioned above informs about the main features to be implemented at a certain point in time. Its elements are loosely defined, and are described by short snippets, 1-10 sentences of texts. Elements are in principle already elicited and prioritized when listed in the roadmap. During a planning meeting held after a busy period with a major deadline, where part of the plan for the future was to be set, the CPO states: "As you can see there's already a roadmap, but the roadmap elements are so widely defined features that we need to refine them before they can serve as tasks", indicating that roadmap to some degree is a work-in progress plan, rather than something written in stone. "Yes, there's a roadmap. But the understanding of the roadmap and its features varies within the team. People may have different understanding of a feature.", states the PM.

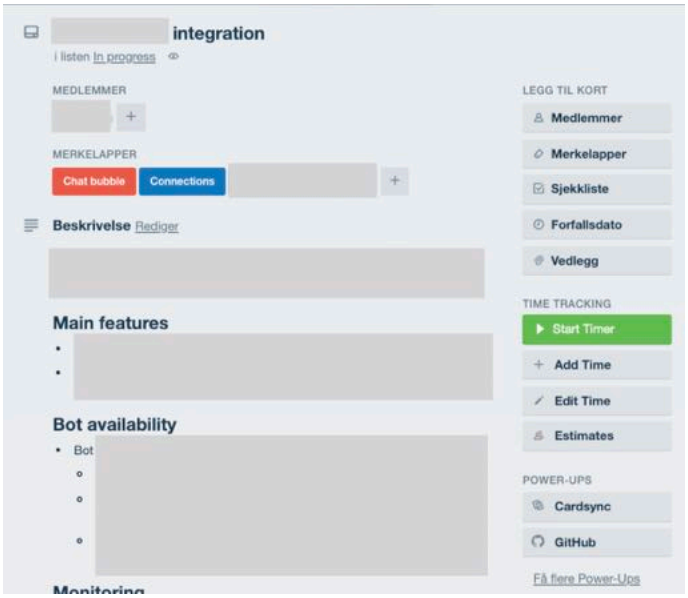

Figure 4.3: Example of Requirement Presentation in Trello

"We've recently introduced a new system for the features to be added, called design documents, which is to be used for features with at least some complexity. It should give an overview and discussion of a feature, including user experience. " states a tech lead. "The design doc should serve as a basis for implementing the feature,

from which we can generate smaller tasks, starting with a minimal viable product idea(MVP)". He continues "the design doc should be a dynamic document, modified along the way". "It involves, like traditional requirements, upfront work. I guess the difference lies in it being flexible, not written in stone from the beginning."

"The motivation of the design documents is to get all facts on the table, make them available easily for everyone, to create a platform on which we can think the feature through(for data structures, user stories, mockups...) before jumping into it. Feedback gathered during the implementation phase may also be put there."

"The user stories are made use of as it provides the right extent of specificity and flexibility - the goals are defined, but it does not specify details on how it should be implemented". "When splitting into smaller tasks in the planning phase, and then doing them one by one during the project, you risk losing an essential view of the bigger picture during the implementation phase."

The change log of the design docs reveals that the design documents are used by a small number of tech team members (observation made approximately a month after they were introduced).

"Creating a 10 page requirements document partly based on assumptions on what the product will be doesn't make sense to me. In my experience it's more efficient to get going right away. Like the period we're in right now: I'm so busy I don't have the time to use or update the design documents".

"For the client project I'm working on now, some documentation and messages are written down in different channels, some are in emails, and some are in the head of people. So it's important that we speak together".

4.2.6 Requirement Estimation

One factor affecting the prioritization of features to be implemented is how much time and/or resources one thinks the task will take, in other words what one *estimates* the task will require of time and resources. During meetings, the term "low-hanging fruit" has repeatedly been mentioned, signifying features which are easy or quick to implement and useful for the users of the product, indicating that estimation is done. So is the case with an attribute to roadmap, elements, done below.

For data collection and analysis, it has been separated between estimation of larger features, typically roadmap elements or larger requests from customers, and estimation of individual tasks, typically a card in Trello.

Estimation of both large and small units of work is to a large extent done informally, based on intuition. However, the company have introduced frameworks for estimation, as explained and discussed here:

Approximately half of the elements on the roadmap are marked with two numbers ranging from 0 to 100 - one reflecting cost, proportionally reflecting the estimated relative time for developing it, the other being the value it will add to the product. By combining the value added and cost factors, one could easily find which features to implement, and which to leave out. "There's not yet a formalized way of how and who decides the cost estimate", states the CPO. Roadmap elements tend to be loosely defined, making estimation more challenging.

Regarding larger features, typically coming from the roadmap, one tech team member states "I typically intuitively know whether something will take one or six months. Often, features including much front-end take much time, with a lot of back-and-forth during the project. Back-end tasks are often more straight-forward.", states a tech team member.

Concerning estimation of (smaller) tasks, a different framework was tried out: Mid 2018, a long debate on how to best estimate the time needed for the work to be done was held, and the team landed on the "moon emoji" system - ranking a Trello card from taking 1 to 5 days. After the system being introduced, it was generally followed and well accepted. The system provides estimation for cards, not features, generally consisting of one or more cards. A few months later, the emojis are not appearing quite that frequently, but are still to some extent made use of.

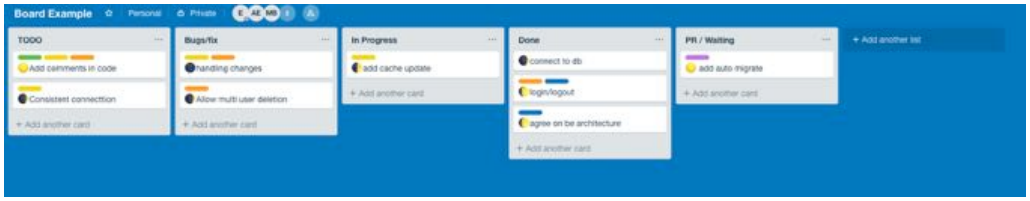

Figure 4.4: Example of Trello Board with Emoji Estimations

The tech team expresses various opinions on systematic estimation, and an overweight of the interviewees express uncertainty of its purpose: "Appreciate estimating, in that it makes me plan the tasks, and split up things if work units are too large. Motivating with many small, rather than one big.", states a junior developer. "But it's difficult to estimate, as even small things often can get more complicated than foreseen".

The emojis added a playful tone to the estimation process, making it feel less heavy: "The emojis are working rather good. Not too stressful, in that I estimate in days, not in hours. It's abstracted away from hours, and I also forget that it is related to days".

Two tech team members expresses uncertainty or disagreement with estimation: "I'm uncertain about this. No examples of negative consequences of not using it - it's useful in some cases, while the product manager wanted to use it for all tasks", states one tech team member. Another tech team member states: "Estimation makes no sense. All my experience has proven that the human is unable to estimate within

precision that makes estimation useful".

4.2.7 Requirement Prioritization

Roadmap elements are prioritized by the CPO together with the leads of each team in the company. When the tech team is ready to implement a new roadmap feature, tech team members pick a new roadmap element. How this new feature is further handled is up to the developer(s) handling the feature.

Roadmap features are often then decomposed into Trello board cards by either the tech leads or the developer being assigned to the task, and generally worked on immediately or shortly after.

The Trello tasks(cards) in the incoming column are handled and prioritized during a weekly tech team meeting. Cards are, depending on the priority and the urgency, either deleted or put in one of three columns, loosely defined as follows:

- Up Next: Top priority, tasks to be done next
- Later: Mid priority - Tasks to be done when "up next" tasks are done
- Planning/Backlog: Tasks to be done at a later point

A developer normally picks tickets from "Up next" when he has finished an earlier task, but this is not a strict rule. If a developer rather wish to pick a ticket from *Later* or *Planning/Backlog*, he is free to do so.

This, together with the freedom during the picking of elements from the roadmap, implies a large degree of freedom, and less top-down control from: "Developers freely chose which tasks to work on and how they want to do it. It should give motivation and creating a feeling of ownership to the product.", explains a tech team member/tech lead.

A result of this pipeline is that tasks stack up in the three columns mentioned above, depending of the load of the incoming column - the developers can only handle so many tickets, whereas the requested work load is unlimited. The team thus regularly reassess the *Later* and *Planning/Backlog* columns, either deleting cards or giving them higher priority, typically moving them into *Up Next* or directly *In Progress*.

"Not implementing all tasks created all tickets related to a feature during the planning or implementation phase of that feature is good - some of them probably proved less important than new tasks who showed up later. Others might have proved irrelevant after feedback from the customer", one tech team member states.

"Some tasks are obviously hot fixes that need to be done right away - if we discover an urgent bug, we won't wait for the weekly meeting to fix it.".

During one meeting, the process of offering a service to the client, a tender, affected the requirement and task prioritization(the tasks in that case being related to scalability) - the alternative making sense from both a technical and product point of view got lower priority than expected, after the PMs information on the tender.

4.2.8 Requirement Validation

Validating requirements in Blueberry most often meant testing early versions, MVPs of a feature, or a the feature after it was considered done.

New features go through a quality assurance process in the "staging" environment of the product. A slack channel for notifying internal users that they should test the new feature provides an easy way for the tech team to test new features. A template for testing instructions is included on each new testing request. This is done as quality assurance, when the feature is assumed to be more or less ready.

When first versions (MVPs) are done, this is tested on internal users of the platform, and occasionally on external, i.e. customers. Some customers get access to a beta-version of the platform, gaining early access to features.

Blueberry get feedback on the product from internal users as well as external users through the customer support channel, i.e. email, and through the product manager, who is in direct contact with the customers as well as in close dialogue with the sales team.

The company otherwise have little formalized processes for testing features, or checking with customers if ideas for features(not yet implemented) would be of value - depending on the feature, the customer, or other factors.

4.2.9 Non-Functional Requirements

One way of categorizing requirements are separating between functional and non-functional requirements. Functional are related to functionality or behaviour, such as "allow users to send personal messages in the platform". The latter are related to the criteria that judge the operation of the system, such as "the server should be able to handle until x requests/second".

Blueberry does not formally distinguish between functional and non functional requirements, FRs and NFRs. However, roadmap features, in principle being the most important tasks, are exclusively functional, potentially leading to a systematic

underpriorization of NFRs. “The features related to the NFRs aren’t much to brag about in front of clients... Or perhaps it is, when I think of it.”, states a tech team member.

Even though NFRs generally do not gain formal priority in the roadmap or in the task management system, they still get it done: “Lately, we’ve organized theme days for non-functional requirements, as these tend not to be prioritized in the daily work. One for UX, one for testing and one for scalability on our platform.”, states a tech team member. Blueberry did not experience any issues with scalability with its product, despite having clients with numerous and frequent users.

“Concerning security - here I suspect that we’ll at some point run into some unpleasant surprises”, states a tech lead. “A lot of security features are stuff you won’t notice that you’re missing. E.g. email notification whenever someone logs onto your account from a new device.”, he continues, suggesting that security often is not considered or prioritized. No security breaches, data leakages or longer periods of down-time were reported, however, indicating that despite somewhat informal approach to security, this lead to no issues.

During the autumn 2018, requirements related to GDPR were elicited, prioritized and implemented. “That kind of pressure forced us do the stuff, so it was actually helpful”.

“There hasn’t been much focus on NFR here. We’re very fortunate to have customers who send us those kinds of requirements - they may seem annoying, but they make us sharper”, states the PM.

One way of prioritizing NFRs have been to organize semi regular workshops themed around such matters. After a theme day on scalability, one client requests data on how many concurrent sessions the platform can handle. The client’s question lead to further investigation, and a load limiter on the API was implemented, reducing the risk for attacks such as DOS (also implying that it earlier had been vulnerable to such attacks).

“A right balance NFR and FR is obviously optimal. A stable product, which proves to be completely useless is obviously not interesting”, states the PM.

Parts of the discussions around scalability and infrastructure are held without the CPO’s or the PMs involvement, thus not becoming a part of the overall plan for the product, such as the roadmap.

The lack of extensive formalized NFRs in the company could potentially lead to technical debt, but data generated from observations and documents did not indicate this being a considerable problem. “It’s difficult to say whether more work earlier would have saved us more time later, but I don’t think it has crossed my mind that it would“, states one developer, and continues “I’m fan of not doing things before it’s actually needed, and I can’t say we’ve experienced major issues or challenges due to

technical debt". The PM states "we haven't suffered much from not prioritizing NFR more earlier." This indicates that technical debt has not been a pain point, despite NFRs not being formally prioritized.

Chapter 5

Discussion

The discussion consists of the findings relevant to a selection of isolated aspects of requirements engineering: Elicitation and prioritization. The discussion on prioritization also contains two sub categories: Those of estimation and non-functional requirements. A discussion of how the aspects of requirements engineering relate to each other, as well as to other aspects of software development and startup engineering follows.

5.1 Elicitation

The flexibility towards work method is also reflected in the requirements elicitation techniques in Blueberry. The requirements elicitation methods are mainly informal, which is common in software startups(Berg et al., 2018).

The case results revealed a number of sources for requirements, and for elicitation of these: The roadmap, provided by managers and tech/design leads, provides the larger features to be implemented. It is not written in stone however, as it was observed that such elements were changed or removed, while according to the time plan was in progress.

A number of case participants report their satisfaction with being less reactive than previously, leaving space for the company's internal long term wishes: "...we now can challenge customers and even turn down requests for features, and I'd say that's a major strength", states one tech team member. This can partly characterized as Blueberry going from *customer-driven* to *market-driven* requirements engineering, partly due to an increasing number of customers and partly due to the company gaining maturity and developing an established plan for the product - a direction for the company to follow in the long term. This comes in addition to the financial

strength and stability gained by the company lately.

The overall plan of the product's future is discussed in company wide meetings. The product vision, and the understanding has gained clarity within in the company, but as of March 2019, three years after the startup being founded, an explicit *product vision statement* was still not stated, being *work in progress*. There is disagreement on the clarity of the product's future, its direction, or the underlying plan for the future - some claim its properly communicated while others experience it otherwise . This could have negative consequences for the company in the phase it's in: The company largely relies on tacit knowledge, and developers currently are given a relatively large degree of freedom for what they want to do: illustrated, amongst other things, by the fact that developers generally pick tasks from the task management platform by themselves. Klotin's conclusion on common requirements challenges in startups confirms Blueberry's challenge: "... the applied requirements engineering practices are often rudimentary and lack alignment with other knowledge areas. As a consequence, inadequacies in requirements engineering hinder other engineering activities and might lead to unwanted technical debt, poor product quality, and wasted resources on building irrelevant features." (Klotins et al., 2019b).

While a challenge frequently encountered amongst startups is difficulties in engaging an early community of engaged customer, providing valuable feedback and requirements validation generating new requirements to be elicited(Klotins et al., 2019b), Blueberry has not lacked a customer and user mass from which it can collect ideas for further product development from.

A factor complicating the company's requirements elicitation is that it is partly operating within a new field of knowledge and technology: *tech-push* and *market-pull* are two main drivers, and Blueberry is considerably driven by the former. Being near the frontier of its domain, the world is unpredictable, hindering long-term planning of the totality of the company's activities.

The company got an increasing awareness of the use of MVPs, and to keep the scope of features down at the beginning. Although it in practice proved to be more difficult define; "What is an MVP, really? I think we lack some reflection on this, and a process of defining this. What the MVP varies from feature to feature, and it's difficult to define." states on developer, and also suggests that this issue has caused a number of features to grow in scope. This clearly resonates with the recurrent challenges found in the study of 88 startups (Klotins et al., 2019a): "A time-consuming and expensive MVP is a sign of poor technology choices and overscoping, the first start-up antipattern.". This can also be seen in light of the findings from (Giardino, Paternoster, et al., 2016): While startups wish to follow the theory from Lean Startup, they find it hard to apply them.

5.2 Prioritization

What determines an item's priority is not only the value it at some point will provide, but also the cost of implementing it, which is found by estimation. Before beginning the discussion of prioritization, we will say a few words about estimation:

Estimation has been found to serve a number purposes: Primarily its role is to allow for the tech team in general and managers specifically to know how much time a task requires. This is done for resource allocation, as well as to inform clients on how much time it takes before a requested feature is delivered, being *prioritization*. Generally, all large features to be implemented are present in the roadmap, even though it could occur that roadmap elements were removed.

Secondly, a developer expressed the advantage of estimation in relation to planning how to work on a task: Being forced to think through what a task would require had a positive impact on the implementation phase.

While serving for different purposes, explicit estimation concretely takes place in two ways: Estimation of defined tasks, ready to be effectuated. These are typically items in the task management systems, generally taking from an hour to a few weeks to implement. This estimation is done sporadically through a moon emoji system: The larger the moon, the more time the task will require, ranging from one to five days.

The second category of estimation concerns larger units of work. A new feature, a major change in the layout, or architectural or infrastructural changes are examples of these. This tasks typically span from a week to several months of work. One tech team member stated "I typically intuitively know whether something will take one or six months", describing such features. Such estimations are in Blueberry generally used for features documented in the road-map, where requirements estimation takes place through the *cost* and *gain* indexes. "There's not yet a formalized way of how and who decides this cost estimate", states the CPO. It is generally determined by management, and resonates with practices found in literature(Melegati et al., 2019): Management determines milestones, whereas developer or product team decides on the task implementation order.

The disagreement concerning estimation observed in the case, as well as a feeling of uncertainty of the purpose of estimating, is nothing new in the industry of software development. Estimation is difficult, and often produce estimates which turn out to be far from the actual resources and time spent. "In order to know which resources we have available in the future, and when features that clients have requested will be ready, we need to estimate", stated one tech team member. However, concerning the estimation of tasks was somewhat incomplete, i.e. occasionally only some of the tasks related to one feature were estimated. When parts of a feature remain unestimated, the feature as a whole is in practice not estimated - and would loose interest for planning and prioritization purposes.

A recurrent issue was that of tasks taking more time than expected, due to changes in requirements, growing scope or other reasons. No temporal limit was set, consequently the incentives to hurry up were fewer. One way to resolve this issue is to introduce the extensively used *sprints*, as used in the Scrum method. Short work cycles with defined goals, providing a framework for reevaluation of the the product and its scope during projects. On the other hand, the absence of defined sprints omits the problems of *Parkinson's law*: "work expands so as to fill the time available for its completion", e.g. when the tasks of a defined sprint are done and the team keeps perfectioning a feature instead of moving on to new and more productive tasks.

While working on a feature, the understanding of the feature may evolve, and hence may the prioritization of the tasks change. "Not implementing all tasks created all tickets related to a feature during the planning or implementation phase of that feature is good - some of them probably proved less important than new tasks who showed up later. Others might have proved irrelevant after feedback from the customer", states a tech team member, when discussing a project which grew out of scope, both in time and product scope. This speaks in favor of more reevaluation of the current work being done - or shorter cycles in an iterative work flow.

Prioritization also takes place during the weekly tech team meeting, when trello tasks are put in columns such as "up next" or "later". In addition, a developer generally chooses which tasks to effectuate himself, picking from any column in the task management system rather than "up next". The developers are hence free to choose, and is given a large degree of freedom, giving both responsibility and power of the product development.

An interviewee speaks of a feature which has been implemented, after being informally prioritized by the tech team. "What are the consequences for later development? And maintenance costs? How will it affect users' overall interaction with the product, currently and in the future? This in my opinion hasn't been thought through". Less planning generally leaves more freedom and responsibility to the developers, calling for empowerment of these. A crucial component of empowerment is the transfer of product relevant knowledge: What is the overall product vision? What are the future plans of the product? What is next year's market segment? In the case, people report differently on what is communicated of the product development, or feature prioritization, relevant information. Parts of the findings indicate some of the knowledge is not sufficiently expressed and communicated throughout the company, given the large degree of developer freedom.

The company makes efforts to share this knowledge, e.g. through company wide meetings where also investors are present, where the future of both the product and the company is discussed. However, some data indicates that parts of the information is not sufficiently communicated: an explicit product vision statement was not communicated as of early 2019. The opportunities of the new technology the company is creating and basing the business case makes the road towards the company's future more difficult to foresee. The customer and end user segment was

not clearly defined as of the end of the data collection period. In that sense Blueberry is, while having a number of stable and satisfied customers, still in a*research phase*.

Parts of the information stemming from other requirements elicitation sources are to some extent not communicated: being competitor analysis, dialogue with managers of clients, or feedback from clients given to sales to the sales team. Whereas some is the knowledge is successfully communicated through the weekly product meeting, meetings between management and tech leads, or other fora, some information does not flow so easily - amongst other thing due to the lack of a formal development method framework, and of written documentation.

Requirements prioritization being a recurrent issue of requirements engineering in agile software development(Heikkila et al., 2015), these issues are not unique to Blueberry.

A part of prioritization is to plan what to is to plan and decide what to make when, and when to release it, i.e. release planning, which will be discussed in the following section.

5.2.1 Release Planning

One point of discussion in Blueberry has been on the dimension of planning vs. feedback based, which is a core element of agile principles e.g. in the agile manifesto with *responding to change over following a plan*. Arguments for planning have been improved predictability for management and thorougher processes of requirements elicitation, documentation and prioritization. The lack of plan lead to the implementation and deployment of features which one tech team member questions with "What are the consequences for later development? ... This in my opinion hasn't been thought through."

The majority of the tech team disagreed with the increased upfront planning, considering that too much of the anticipated details would later, during development time, prove irrelevant. As one developer states it "time was spent in management planning upfront and detailing requirements, time that could have been used implementing the stuff. The detailed requirements were partly useless, as they further down the road could prove irrelevant".

The disagreement partly stems from the different viewpoints of the roles within the tech team. Some are concerned with customers and the customers' wishes for product delivery, implying a wish for a degree of predictability. Others mainly see the disadvantages of upfront planning, such as the parts of it which end up proving irrelevant.

Planning releases ahead requires that there will be something valuable to the customer/end user release. In the example of the core technology team, working on new

technologies, this proved more complicated than in the case of the rest of the tech team, who generally work on conventional full stack web development. Cutting-edge technologies are unpredictable, and the work may be characterized as *research*, or experimentation. The difficulties associated with less predictability raises concern amongst one tech team member: It's challenging to manage, and the effect of some degree of lack of timeboxing, i.e. working without a fixed temporal framework, can have negative effects on development speed. Seen in perspective, the core technology work is not an isolated and extreme example of the craft software development, but rather a interesting example of software development in a startup environment, with the property of *unpredictability* being more explicit than normally. Ries characterizes startups as operating under extreme uncertainy(Ries et al., 2012) well describes well the company at its birth, but as of early 2019 the company is generally stable in terms of product, customers and hence income. However, the core technology activities still experience the uncertainty and unpredictability of startup.

5.2.2 Non Functional Requirements

An own category of requirements to be taken into account when being handled are non-functional requirements: Non-functional requirements (NFR) is in literature presented as one of the challenges of agile requirements engineering, partly due to the extensively used user stories(Inayat et al., 2015). Also in Blueberry, NFRs have been given less explicit priority than functional, with the exception of user experience due to active designers and front-end developers. Some tech team member are concerned with and work for an improved security of the company's applications, but it is security features are rarely explicitly prioritized and planned for. One (perhaps pessimistic) tech team member states "Concerning security - here I suspect that we'll at some point run into some unpleasant surprises", suggesting that lack of focus on security will bite them in the back.

Architectural decisions, and questions of how much load the the main application should be able to handle, are also expressed in a lesser degree than "normal features", to some extent compensated for by regular workshops themed around such matters. These are theme days primarily focused on specific features. The workshop's effect was reported to be varied: Some workshops result in valuable change, while others provide less permanent effects.

One might intuitively suspect that the lack if focus and prioritization on NFRs lead to trouble, and *technical debt*, down the road, but the observations prove otherwise. No major issues related to tech debt have been observed. Concerning scalability, no severe issues have been reported, despite clients with large amounts of users. The same goes for security issues, data breaches or down time of essenial applicacations. This confirms the "Startups reward hacky code"(Dalal, 2019), and one may question whether Blueberry should, at some occasions, produce "hackier code", meaning producing quicker at the price of lower quality. Faster development time would

allow for quicker feedback, potentially providing information on which features to developer further and which ones to drop. Insight in and consciousness on which parts of Blueberry's product need to be solid, and which part allow hackier code, would in this case be necessary.

5.3 Dependencies and Methodology

Characterizing and framing Blueberry's software development is challenging. Blueberry is a young company still in the process of establishing its ways of working, making it more difficult to categorize and characterize. That Blueberry changes its work methods as the number of clients and employees keep increasing is a finding of its own: it lays in its dynamics and flexibility rather than in static methods, and Blueberry is rightly doing so. This is backed up by the relative inexperience of the tech team members, with a slight majority of the employees having little or no prior work experience before entering the company. This is in line with findings from literature: Due to the changing and uncertain environments, their small size and the *need for speed*, software startups are reactive in their processes(Berg et al., 2018).

On the other hand, the number of tech team members remained the same from September 2018 to April 2019, without landing on a well defined work flow - Blueberry has struggled in finding its optimal ways of working. "... we try out new methods until landing on one method that will accepted for a longer period of time and in a number of contexts.", states a tech team member, reflecting both that the context is changing, and that Blueberry has struggled to establish an underlying working framework that is accepted by and well functioning with the team. One example of this is the upfront requirements documentation in the task management system which were generally not accepted, followed by the dynamic design documents being introduced by the end of the data collection period, not immediately adopted by the tech team.

The variability in work methods, as well as a lack of rigour, requires a larger degree of tacit knowledge within the team than if the work flow was formalized, exemplified through the statement of a developer, concerning the documentation and knowledge sharing of a project: "... some informations are written down in different channels, some are in the head of people...". As found in (Klotins et al., 2019b), where the technical debt of the investigated cases only became problematic only after hiring new people, the lack of an established work flow may become especially problematic in relation to new hires. It can also prove more challenging in engaging junior developers: It was observed that the junior developers were generally less participating during meetings, indicating less engagement in process matters.

As seen in the literature chapter, there exists a number of framework for categorizing or determining which phase a startup is in(Tripathi et al., 2018). For context, it's worth placing Blueberry in the frameworks. This provides insight in which challenges

startups in similar phases have encountered, and how to solve them.

In Crowne's four phase model(Crowne, 2002), Blueberry can be sorted and placed as being in the third phase, *growth*: "This phase begins when the product can be commissioned for a new customer without creating any overhead on the development team. It ends when market size, share and growth rate have been established and all business processes necessary to support product development and sales are in place". Blueberry appears to be encountering some of the difficulties described by Crowne, associated with the *growth* stage, but also challenges related to Crowne's other stages, as described in the theory chapter. One challenge is that of the lack of a "robust process to capture product ideas, develop them and choose between competing ideas". Blueberry's challenge does not lay in the lack of ideas for further development, but rather a framework for handling them: parts of activities concerning documentation, prioritization and validation is to some extent done ad-hock, rather than following a strict framework. This comes at the price of potential suboptimal product development, as well as disagreements, as some matters do not get thoroughly discussed throughout the company.

When building new products, Blank suggests that startups should focus on finding the problem/solution fit through investigation with customers and/or users and prototyping before finding the right product/market fit(Blank, 2003). Blueberry has had a number of paying and satisfied customer, and a number which keeps increasing. It has hence found a good problem/solution fit, and has largely also found its product/market fit. Startups are generally not offering one single and atomic product, however. Blueberry has numerous possible features which may be implemented, and it has challenges finding the right problem/solution fit for the additional features even though it has proceeded correctly with its initial product. The problem/solution and product/market fit is hence an unuseful framework for mature startups like Blueberry.

Blueberry passed a turning point during summer/fall 2018, when doubling the number of members in the tech team. With a new size of 15 technologists, and the proportion of employees with previous professional experience somewhat lower than previously, the team needed more structure and formalisms than earlier. A product manager was hired in the same period, amongst other things being responsible for the necessary changes. Some of the changes generally met resistance, such as the increase in upfront planning and systematic task estimation. The resistance can partly be explained by, and seen in context of the original team having its old and successful ways of working, potentially having an inherent resistance towards change. The change resistance is enforced by the culture in the software industry calling for flat hierarchies and more freedom to the ones effectuating the work - the developers. Developers may disagree with the increased effort in process matter, resulting in less time spent producing working software.

In Blueberry these phases often take place rather informally, without a strict frame around the activity. With a stricter framing, it would from a research point of

view be easier to distinguish and analyse the separate phases, such as *elicitation*, *documentation*, *estimation*, *prioritization*.

As seen in literature, it's common to decompose the requirements engineering process into separate phases such as the ones mentioned above(Tripathi et al., 2018). It makes sense to split the process into logical units of activities, as it makes analysis and discussions of the field more constructive. However, these components remain interdependent of each other, and one risks losing important insight in the matter when considering the components isolated. This speaks in favor of considering requirements engineering holistically, or investigating *requirements as a process*. Whereas one in literature can read about the set of discrete constituents of requirements engineering, also in that related software startups, the observations proved otherwise. The process of requirements engineering also was dependent on and affected other aspects of software development methods, such as release planning. During the requirements elicitation, there is already aspect of both estimation, through an initial impression of the effort needed for the task to be done, and prioritization, as a number of additional features to the product would simply not be taken into account due to their irrelevance. Typically during iterative processes, requirements are elicited, documented, prioritized, during each iteration, more or less formally. Requirements validation, typically happening after the implementation of a prototype or an early version might reveal bugs, which again generate tasks who are effectuated while skipping a number of stages of a traditional requirements engineering.

5.4 Terminology

One way of describing Blueberry's software development is determining to which degree, and along which dimensions its software development can be categorized as *agile*. Determining whether or not Blueberry can be characterized as agile in general proved difficult, as well as having little interest. Views on agile software development(ASD) were differing and generally showed that the term was so too wide to make useful sense, indicated in quotes like "I hardly dare to use the word". This resonates with what was found by (Conboy, 2009), indicating that ASD has become such a fuzzy subject that it has become difficult to discuss it. Combined with its massive adoption in industry(State-of-Agile, 2018), this indicates that ASD as of 2019 is difficult to express clearly, perhaps becoming more of a underlying culture than clearly stated principles, and that the phenomenon is *hyped*. This calls for more clarity when discussing agile methods in industry, in contrast to vague talk about the matter, which seems to be common. A solution is to consider separate aspects of ASD rather than the phenomenon holistically. The subject of discussion should rather be the the distinct aspects of software development, rather than *agile software development* in general.

Requirements engineering in the field of software is a somewhat misleading term.

Requirements is used in a broad sense and roughly includes all potential tasks and ideas for what is to be developed. Effectuated before implementation phase, it may be seen as more of a management activity than related to what one associates with *engineering*. The bad term, or *misnomer*, causes confusion when communicating the knowledge area, and I propose *Software Activity Handling* as the knowledge area's new name.

5.5 Research Limitations

Descriptions of what has occurred during the case, collected through interviews, may vary depending on the interviewee. The perceptions of reality vary according to the individuals being interviewed. Another factor complicating the data collection and following analysis is that what people *say* may differ from what they *do*. During interviews, interviewees will have an inclination to present own actions as better than what might have been the case. This creates a number of differing versions of reality, as well as a tendency of prettified versions of own actions to be given during given interviews. This occured at a number of occasions during interviews: If not conflicting, fundamentally different versions of something was provided. The different opinions on upfront planning and requirements documentation was an example of this: the manager had a fundamentally different view compared to that of a number of tech team members. Observations of meetings and documents play an important role in adjusting for these differences, which need to be taken into account before the analysis. In hindsight the data collection has been too broad. Much time was spent collecting data which was not relevant to the research theme. This can be explained by the lack of research experience by the researcher, as well as a a continuous exploration and improvement of the field.

Although not a traditional field of knowledge within software development, and not discussed in this paper, group dynamics affect the case results. How developers, designers and managers interact affects how they work together, and consequently how they act within the developer team. An especially fruitful cooperation between two individuals positively impacting the work flow, or on the contrary, an unreported conflict, would affect aspects of work flow investigated in this research project. This poses generally a larger threat to single case studies like this one than in multicase studies, where each individual, or data point, affects the case results more.

Chapter 6

Conclusion

The study has revealed that analyzing requirement engineering by considering static and independent activities like elicitation, documentation, estimation, prioritization and validation has its limitations. A more holistic approach will add more to the research field: The dependencies between these aspects, as well as with other knowledge fields related to requirements engineering like release frequency, iteration frequency, degree of upfront planning and management style are considerable and must be taken into account. The requirements engineering activities taking place can be characterized as being a process.

Giving product development maximum *speed* on one hand, and achieving correctness on the other, are conflicting interests . A developer-first approach, giving those who build the power to work with limited management overhead comes at the risk of having great speed *in the wrong direction*. Developer freedom, allowing for quick and reactive development requires a high degree of tacit knowledge. In order to do so, companies need to focus on systematic information sharing, and perhaps implement frameworks for such. The information stemming from the sources identified in requirement elicitation or sources need to be spread, as well as providing awareness of the sources being used.

Software startups tend to follow agile principles, as they rely on their reactivity rather than on upfront planning. However, as both criticized by case participants and observed in industry, there's a tendency to mix up *agile* and *chaos* - the lack of systematic processes and deliberate and explicit choices around these are being excused by (ab)using the *agile label*. The distinction between these seems difficult to make, perhaps due to the phenomenon being hyped.

Software startups are not agile by nature. But they share common characteristics, especially those stemming from the shared interest in speed, flexibility and reactivity. In addition to the growing amount of literature from the field of startup engineering, findings from agile software development can also apply for the knowledge area of

startup engineering.

Requirements engineering is a bad term, a *misnomer*, for describing the knowledge area it encompasses and creates unnecessary confusion, especially while communicating towards. I propose *software activity handling* as the new term.

6.1 Further Research

Software startups are continuing to mark society, and continues to gain presence and popularity(TU, 2016). Research has been effectuated, some in the beginning of the century, an increasing effort is done(Startup-Organization, 2019) by the research community. The knowledge area is gaining maturity, but a lot remains to be done - for companies like Blueberry, many questions remain unanswered and further guidance in the unpredictable and chaotic environment they operate in is advantageous.

While this research paper has focused on *requirement as a process*, including the dependence of the components constituting the field of requirements engineering, one point of learning is that the breadth of the research has been made at the cost of depth: Further literature research, data collection and analysis can be made in a number of fields. Aspects of requirement elicitation such as requirement sources or selection, and techniques for this have to some degree been investigated(Tripathi et al., 2018), but need further investigation. The usage and consequences of non-functional requirements in software startups seems largely undiscovered and undocumented, as well as the potential link to technical debt.

All of the above must be done while keeping findings from both startup and agile software engineering as backdrops, for different life cycle stages of startups. This will help startups in specific context gain relevant knowledge.

References

Abrahamsson, Pekka et al. (2017). "Agile Software Development Methods: Review and Analysis". In:

Agile Manifesto (2001). URL: www.agilemanifesto.org.

Agile Software Development: Current Research and Future Directions (2010). eng. Springer Berlin Heidelberg: Berlin, Heidelberg. ISBN: 9783642125744.

Aoyama, M. (1998). "Agile Software Process and its experience". eng. In: IEEE Publishing, pp. 3–12. ISBN: 0818683686.

Benington, Herbert D. (1983). "Production of Large Computer Programs". eng. In: *Annals of the History of Computing* 5(4), pp. 350–361. ISSN: 0164-1239.

Berg, Vebjørn et al. (2018). "Software startup engineering: A systematic mapping study". eng. In: *The Journal of Systems and Software* 144, pp. 255–274. ISSN: 0164-1212.

Blank, S. (2003). *The Four Steps to the Epiphany: Successful Strategies for Products that Win*. Lulu Enterprises Incorporated. ISBN: 9781411601727. URL: https://books.google.no/books?id=oLL2pjn2RV0C.

Boehm, B.W. (1983). "The Hardware-Software Cost Ratio: Is It a Myth?" In: *Computer* 16(3). ISSN: 0018-9162.

Conboy, Kieran (2009). "Agility from First Principles: Reconstructing the Concept of Agility in Information Systems Development". eng. In: *Information Systems Research* 20(3), pp. 329–354, 478. ISSN: 10477047. URL: http://search.proquest.com/docview/208160739/.

Creating the agile library : a management guide for librarians (1998). eng. Westport, Conn.

Crowne, M (2002). "Why software product startups fail and what to do about it. Evolution of software product development in startup companies". eng. In: *IEEE International Engineering Management Conference*. Vol. 1. IEEE, 338–343 vol.1. ISBN: 0780373855.

D. Linthicum, TechBeacon (2016). *Back to waterfall: When agile and DevOps don't work well*. URL: https://techbeacon.com/back-waterfall-when-agile-devops-dont-work-well.

Dalal, N (2019). In: URL: https://hackernoon.com/four-startup-engineering-killers-1fb5c498391d.

Dove, R. (1993). *Agile Defined*. URL: http://www.parshift.com/Files/PsiDocs/Pap930701Dove-BeginningTheAgileJourney-A%20Hewlett%20Packard%20Guidebook.pdf.

Giardino, Carmine, Nicolo Paternoster, et al. (2016). "Software Development in Startup Companies: The Greenfield Startup Model". eng. In: *IEEE Transactions on Software Engineering* 42(6), pp. 585–604. ISSN: 0098-5589.

Giardino, Carmine, Xiaofeng Wang, and Pekka Abrahamsson (2017). "Why Early-Stage Software Startups Fail: A Behavioral Framework". In:

Gralha, Catarina et al. (2018). "The Evolution of Requirements Practices in Software Startups". eng. In: *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. ACM, pp. 823–833. ISBN: 9781450356381.

Heikkila, Ville T et al. (2015). "A Mapping Study on Requirements Engineering in Agile Software Development". eng. In: *2015 41st Euromicro Conference on Software Engineering and Advanced Applications*. IEEE, pp. 199–207. ISBN: 9781467375856.

Inayat, Irum et al. (2015). "A systematic literature review on agile requirements engineering practices and challenges". eng. In: *Computers in Human Behavior* 51(PB), pp. 915–929. ISSN: 0747-5632.

Jorgensen, Magne (2016). "A survey on the characteristics of projects with success in delivering client benefits". eng. In: *Information and Software Technology* 78(C), pp. 83–94. ISSN: 0950-5849.

Karlsson, Lena et al. (2007). "Requirements engineering challenges in market-driven software development – An interview study with practitioners". eng. In: *Information and Software Technology* 49(6), pp. 588–604. ISSN: 0950-5849.

Klotins, Eriks, Michael Unterkalmsteiner, and Tony Gorschek (2019a). "Software Engineering Antipatterns in Start-Ups". eng. In: 36(2), pp. 118–126. ISSN: 0740-7459.

Klotins, Eriks, Michael Unterkalmsteiner, and Tony Gorschek (2019b). "Software engineering in start-up companies: An analysis of 88 experience reports". eng. In: *Empirical Software Engineering* 24(1), pp. 68–102. ISSN: 1382-3256.

Medeiros, Juliana et al. (2017). "An Approach Based on Design Practices to Specify Requirements in Agile Projects". In: *Proceedings of the Symposium on Applied Computing*. SAC '17. ACM: Marrakech, Morocco, pp. 1114–1121. ISBN: 978-1-4503-4486-9. DOI: 10.1145/3019612.3019753. URL: http://doi.acm.org/10.1145/3019612.3019753.

Melegati, Jorge et al. (2019). "A model of requirements engineering in software startups". In: *Information and Software Technology* 109, pp. 92–107. ISSN: 0950-5849.

Nagel, Roger N (1992). *21ST Century Manufacturing Enterprise Strategy Report*. Agile Mfg Enterprise Forum Bethlehem Pa. URL: http://handle.dtic.mil/100.2/ADA257032.

Nguyen, Anh, Pertti Seppänen, and Pekka Abrahamsson (2015). "Hunter-gatherer cycle: a conceptual model of the evolution of software startups". In: pp. 199–203. DOI: 10.1145/2785592.2795368.

Norsk senter for forskningsdata (n.d.). URL: www.nsd.no.

Pantiuchina, Jevgenija et al. (2017). "Are Software Startups Applying Agile Practices? The State of the Practice from a Large Survey". In: *Agile Processes in Software Engineering and Extreme Programming*. Ed. by Hubert Baumeister, Horst Lichter, and Matthias Riebisch. Springer International Publishing: Cham, pp. 167–183. ISBN: 978-3-319-57633-6.

Rafiq, U. et al. (2017). "Requirements Elicitation Techniques Applied in Software Startups". In: *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pp. 141–144. DOI: 10.1109/SEAA.2017.73.

Regnell (2005). "Market-Driven Requirements Engineering for Software Products". eng. In: *Engineering and Managing Software Requirements*. Springer Berlin Heidelberg: Berlin, Heidelberg, pp. 287–308. ISBN: 9783540250432.

Ries, Eric and RIES ERIC (2012). *Lean Startup*. eng. Village mondial. 0. Pearson France. ISBN: 2744065080.

Royce, W (1987). "Managing the development of large software systems: concepts and techniques". eng. In: *Proceedings of the 9th international conference on software engineering*. ICSE '87. IEEE Computer Society Press, pp. 328–338. ISBN: 0897912160.

Startup-Organization (2019). In: URL: https://softwarestartups.org/.

State-of-Agile (2018). *State of Agile 12th Edition*. URL: https://explore.versionone.com/state-of-agile/versionone-12th-annual-state-of-agile-report.

Sutherland Takeuchi, Rigby (2016). *Embracing Agile*. URL: https://hbr.org/2016/05/embracing-agile.

SWEBOK 2004 (2005). eng. Place of publication not identified: Institute of Electrical and Electronics Engineers, Inc. Staff.

Tracz, Will (2015). "Agile: The Good, the Hype and the Ugly, Written by Bertrand Meyer". eng. In: *ACM SIGSOFT Software Engineering Notes* 40(2), pp. 38–39. ISSN: 0163-5948.

Tripathi, Nirnaya et al. (2018). "An anatomy of requirements engineering in software startups using multi-vocal literature and case survey". eng. In: *Journal Of Systems And Software* 146, pp. 130–151. ISSN: 0164-1212.

TU (2016). In: URL: https://www.tu.no/artikler/enorm-okning-i-startups-flere-ser-at-de-ma-skape-sin-egen-arbeidsplass/346300.

YIN, ROBERT K (2011). "Case Study Research". English. In: *Modern Language Journal* 95. ISSN: 0026-7902.