MASTER'S THESIS | LUND UNIVERSITY 2016

Requirements Engineering in Startups with Open Source Software Related Business Strategies

Billy Johansson, Martin Lichstam

Department of Computer Science
Faculty of Engineering LTH

ISSN 1650-2884
LU-CS-EX 2016-38

Requirements Engineering in Startups with Open Source Software Related Business Strategies

Billy Johansson, ama10bjo@student.lu.se
Martin Lichstam, martin.lichstam@gmail.com

September 29, 2016

Acknowledgements

We would like to thank Peter Neubauer, Christoffer Richardsson and Emil Sjödin for devoting their time to participate in our interview sessions. We would also like to thank Johan Linåker and Björn Regnell for their guidance and supervision throughout the thesis process.

Abstract

The role of startups in today's economy grows increasingly more important while more and more companies engage in open source software (OSS) development. Both startups and open source projects are sources of, and rely on, innovation. As startups are typically very resource constrained and open source potentially offers low cost labour and innovation sources; the intersection of startups and OSS is worth studying.

In this thesis we perform a literature study on the current state of Requirements Engineering (RE) in startups and OSS, showing that the RE process is comparatively more informal in both startups and OSS than what would be the norm in classical RE. We also present interviews with four different startups and examine how they manage their OSS projects through an RE perspective confirming the results from the literature study. In addition we identify a set of RE related challenges faced by the interviewed startups when managing their OSS projects and present potential actions that can be deployed to alleviate the challenges. To further aid the startups in this area we identify four different themes related to the challenges that indicates broader issues in a startups OSS RE bridging process and present a method for identifying areas of improvement in a startup's ongoing OSS project.

Contents

1 Background **1**

2 Problem Description **2**

3 Methodology **4**

4 Frame of Reference **7**
- 4.1 Startups . . . 7
  - 4.1.1 Lean Start-up . . . 7
- 4.2 Open Source . . . 8
  - 4.2.1 OS as Part of a Business Model . . . 8
  - 4.2.2 Open Source Licenses . . . 8
  - 4.2.3 Open Source Community . . . 9
- 4.3 RE in Startups and Lean Start-up . . . 12
  - 4.3.1 Requirement Specification . . . 12
  - 4.3.2 Elicitation . . . 12
  - 4.3.3 Release Planning and Validation . . . 13
- 4.4 RE in OS community . . . 14
  - 4.4.1 Elicitation . . . 14
  - 4.4.2 Requirement Specification . . . 14
  - 4.4.3 Release Planning and Validation . . . 15
  - 4.4.4 What to Reveal . . . 15

5 The Startups **16**
- 5.1 Mapillary . . . 16
  - 5.1.1 Business Model . . . 16
  - 5.1.2 OSS Strategy . . . 17
- 5.2 Neo Technology . . . 18
  - 5.2.1 Business Model . . . 18
  - 5.2.2 OSS Strategy . . . 18
- 5.3 Bitcraze . . . 20
  - 5.3.1 Business Model . . . 20
  - 5.3.2 OSS Strategy . . . 20
- 5.4 RefinedWiki . . . 22
  - 5.4.1 Business Model . . . 22
  - 5.4.2 OSS Strategy . . . 22
- 5.5 License and Legalities . . . 24

6 The Internal and External RE Process **25**
- 6.1 Elicitation . . . 25
- 6.2 Requirements Specification . . . 27
- 6.3 Validation . . . 30
- 6.4 Release Management . . . 32

7 Analysis **33**
7.1 The Incentives . . . 33
7.2 The Startups Summarised . . . 35
7.2.1 Bitcraze . . . 35
7.2.2 Mapillary . . . 35
7.2.3 Neo Technology . . . 35
7.2.4 RefinedWiki . . . 36
7.3 What to Reveal . . . 37
7.4 The Research Questions . . . 39
7.5 RQ1: Requirements Engineering . . . 39
7.6 RQ2: Challenges . . . 42
7.6.1 Potential Actions to Challenges . . . 43
7.6.2 Themes . . . 46
7.7 RQ3: Engineering Solution Design of BOSS . . . 48
7.7.1 BOSS Revealing Guidelines . . . 49
7.7.2 BOSS Challenge Analysis . . . 51
7.7.3 BOSS Macro Analysis . . . 52
7.7.4 BOSS Action Application . . . 53

8 Validation **56**

9 Threats to Validity **60**
9.1 Internal Validity . . . 60
9.2 External Validity . . . 60
9.3 Construct Validity . . . 60
9.4 Reliability . . . 60

10 Discussion **61**

11 Conclusions **64**

Appendices **65**

A Screenshots of validation webpage **65**

1 Background

In today's economy startups have an essential role for a country of being able to compete on a global scale. The entrepreneurial spirit combined with the agility – which large corporations often seem to be lacking – enables innovation to take place. Many corporations are attempting to leverage on this and as an example, companies like Facebook, Tesla Motors, Apple and Google are intentionally trying to keep their organisations as much startup as possible [28].

Research shows that startups often are characterised by newly created companies with inexperienced teams that operate in a volatile and highly uncertain market – thus making flexibility and adaptation two very important factors. Empowerment of team members is another crucial element due to the time pressure and lack of resources, which consequently forces startups to keep management and administration at minimum expense [10].

The ultimate goal of a startup is to find a viable and scalable business model, hence, one of many possible definitions could be: "*a temporary organisation designed to search for a repeatable and scalable business model.*". Typical characteristics are: lack of resources, highly reactive and innovation (ibid.).

Open Source Software (OSS) is software that anyone can contribute to. It is usually released for free under various licenses that may allow commercial use. OSS projects can emerge from the need of a single developer or be released by a company as part of a business strategy.

OSS is becoming increasingly more utilised as part of a business strategy for startups. This includes incorporation of existing OSS projects in products as well as releasing internally produced intellectual property to the open source community. As an example, OSS could be used to create services and products at low cost – something that would otherwise require extensive capital investment. Another approach is the dual-licensing strategy, where the startup releases parts of its software as open source, in order to benefit from the community and to eventually enhance the proprietary, closed sourced version of their product.

2 Problem Description

From a Requirements Engineering (RE) perspective, the inclusion of OSS in a business strategy has many issues; including (but not limited to) when to include, when to release, what and to what extent contributions should be made, managing OSS stakeholders and steering an OSS project in a direction valuable to the company.

In OSS projects, classical RE specifications are almost completely absent. Instead, the requirements are expressed in an informal way and often in relation to older versions or other competing products. They are found scattered across work products such as issue trackers, emails and How To-guides [24].

Due to the startup circumstances mentioned earlier in the background section, they are forced to adapt quickly to a highly uncertain and rapidly changing market driven environment. This seems to rule out the practicality of classical RE specifications which are therefore hardly used in this context. The customers are rarely known and there might not even be a market for the product, hence, requirements change rapidly. Some startups have adopted a lean approach with heavy focus on quickly eliciting and validating requirements, a method that is gaining more traction [21].

In this thesis the intersection of startups and OSS in an RE context will be investigated. Currently, the literature on the startup subject is somewhat limited and close to non-existent for the parts combined. Fortunately OSS is a more studied field with more material accessible.

The ultimate goal of the thesis is to establish a set of RE guidelines for how startups engaged in OSS development should structure their internal, as well as external and bridging RE practices towards OSS communities to maximise value capture and creation in regards to the startup's business and product strategies.

RQ1: How is RE currently structured and practiced in startups engaged in OSS development?

RQ2: What challenges can be identified in regards to current RE structure and practices in startups engaged in OSS development?

RQ3: How can the problems identified in RQ2 be mitigated?

From the literature study we find that RE practices in OSS projects as well as startups are done very informally when compared to classical RE. The results are mirrored in the interviews and in addition we find a set of RE related challenges faced by the interviewed startups. We propose potential actions to alleviate the challenges and also bring attention to four different themes that indicate broader issues in a startup's OSS RE bridging process and present a method for identifying challenges related to the themes in an ongoing OSS project.

The thesis is structured as followed. Section 3 presents the methodology used and is followed by the literature study presented as a frame of reference in section 4. The frame of reference briefly introduces startups and OSS as a concept and then examines previous research on how RE is done separately in startups and OSS communities. In section 5 the interviewed startups are presented along with their business models and OSS strategies. Following the introduction of the startups we present the results from the interviews in section

7.1 by examining the different incentives the startups had for releasing an OSS project, what they have released and how they manage the projects from an RE perspective. In section 7 the results of the literature study and interviews are analysed and the research questions answered. We also present a checklist and a flow chart that can be used by startups releasing an OSS project. In section 10 and 11 the results are discussed and summarised.

3 Methodology

This section describes the scientific approach of the thesis in two steps – first by providing an overview and secondly by specifying each step of the process.

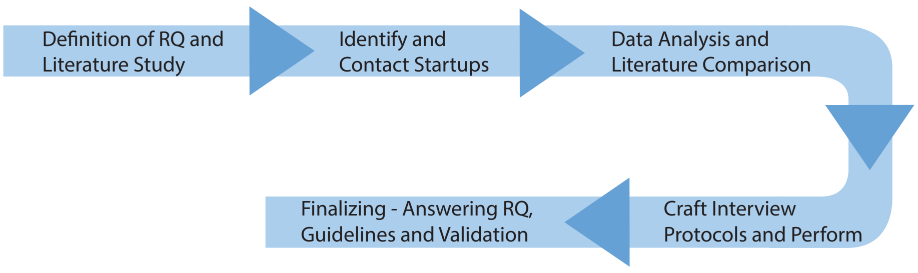

Definition of RQ and Literature Study
Identify and Contact Startups
Data Analysis and Literature Comparison
Craft Interview Protocols and Perform
Finalizing - Answering RQ, Guidelines and Validation

Figure 1: The Process - Overview

Definition of RQ and Literature Study The research begins with a literature study to map the current state of RE in startups and OSS projects separately in an attempt to deeper understand the intersection between them. The material was obtained through Google Scholar, IEEE, Lund University's internal portal and our supervisor. The research questions were crafted with the intention to fulfil certain purposes, in order to be able to derive adequate conclusions and results.

The objective with *RQ1* is to map the current state of how RE is practiced in connection to OSS communities. More specifically, how the startup communicates and coordinates the work, both externally and internally with their community. How are ideas created and what way do the startup pursue to create the final product? This involves everything from potential brainstorming to how the requirements are specified and written down to the development stage – how tasks are noted and how validation is performed. Also, what communication channels are used, both internally and with the community to elicit, validate and specify requirements as well as develop the product. Moreover, it enables an exploration of the general processes, hence, providing a deeper insight which in turn leads to *RQ2* – what challenges can be identified.

RQ2's objective is to pinpoint issues a startup may be facing regarding RE structure and practices. This investigation will provide a useful overview as well as understanding in order to eventually be able to answer *RQ3*, namely, how to mitigate these challenges found in the previous research question.

The answer of *RQ3* will serve as a base for the creation of the final guidelines/framework.

Identify and Contact Startups A case study of an exploratory nature was performed. Participating startups were the following:

Mapillary

RefinedWiki

Neo Technology

Bitcraze

The startups have been carefully selected with the major requirement of OS community involvement-criterion in mind. A more extensive introduction of the interviewees as well as the startups is provided in section 5. The companies were initially found by analysing Malmö's startup scene[1], information that is publicly available on the web. Secondly, an in-depth analysis was performed in order to assess whether the particular startup had an OSS involvement. A list of four potential candidates was created and an email was sent to their representatives that could be found on each startup's website. All startups responded positively to the request for participating in an interview.

The startups were provided with the information regarding the interview's nature, which they subsequently used to select an appropriate interviewee.

Craft Interview Protocols and Perform The interview questions were prepared in advance; however, they served as a guideline rather than a manuscript – a semi-structured interview. The questions were based on the previous step, the literature study, and on the author's empirical knowledge with the main goal of answering *RQ1*.

The interviews were performed face to face, at the startups' offices, recorded digitally and transcribed in its entirety.

Data Analysis and Literature Comparison The analysis was tightly connected with the answering of RQs, however, sections such as *What to Reveal* was conducted separately. This could be seen as a by-product mainly extracted from the interviews. It should be mentioned that the literature turned out to be especially limited and rarely delves into specifics on RE in the OSS-Startup intersection. However, efforts were made to attain information that could be valuable in order to successfully derive a framework. This was conducted by iteratively comparing against the results obtained from the interviews. Also, keywords that were obtained from the interviews were used for literature searching and vice versa. In order to be able to fully analyse the data obtained from the startups, the various parts of the interviews were categorised into the following themes:

Elicitation

Requirements Specifications

Validation

Release Management

OSS

General process

The interviews were conducted once for each startup and transcribed in their entirety. The transcription process offered an extensive learning experience and served as firm support for continued analysis. The amount of material

[1]`http://www.malmostartups.com/`

from the interviews was considerable and allowed several iterations between the literature and the assessed data to take place. These cycles, apart from the study itself, included plenty of discussions and brainstorming between the authors, leading to the birth of several hypotheses which served as a foundation for the next iteration cycle where both the literature and the data from the interview were revisited. Although the hypotheses were not explicitly written down, they worked as an informal road map and played a major role in the development of the cornerstones that eventually shaped this thesis.

All quotes, both in-line and as separate paragraphs in this thesis are, unless nothing else is mentioned, raw data from the interviews. The following are examples of how they may look like:

(. . .) reasoning that *'. . . it's useful for other third parties and they can help us with development.'* while also noting that there was *'no product value for us'* meaning

or

> *What's most interesting for an open source project are things that are reusable, general and solve problems that a large enough group of people have.*

Generally, the startups disclosed an unambiguous view of their current situation, thus making it simpler to identify both potential challenges and potential actions. Furthermore, some startups not only exhibited enormous amount of experience but an outstanding track record as well, implicitly hinting what kind of potential actions that can be used.

Finalisation - Answering RQs, Guidelines and Validation The first research question *RQ1* was answered by summarising the results obtained from the interviews. In other words, providing an overview of the information obtained from each startup.

The second research question *RQ2* was answered by carefully analysing the output of the interviews. The challenges were structured and specified in clear components allowing for further examination of how to mitigate them.

RQ3 was answered by a meticulous comparison between the literature study and the data acquired from the startups. Each challenge identified in *RQ2* was mapped to a potential action, *PA*.

The answers above subsequently lead to the formation of the checklist and the flow chart. These tools were derived by intensive collaboration between the authors as well as with help of inputs from the supervisors.

Validation was conducted by creating a website which was sent out to the four interviewees. This is further described in section 8.

4 Frame of Reference

This section provides a theoretical foundation of the content that this thesis brings into focus. The section starts by introducing the startup and lean startup concepts followed by an explanation of open source and how it can be incorporated in a business model. Different licenses, aspects of a community and preconditions for a successful OSS project are also examined. In sections 4.3 and 4.4 research on how the RE processes of elicitation, specification, validation and release planning are done in startups and OSS is presented. The OSS section also includes research on what type of software is suitable for open source.

4.1 Startups

It is important to recognise the differences between a newly founded company and a startup. The former mentioned often already has a certain path to follow while the latter seeks to explore new business opportunities, thus lacking a definite plan. This journey requires extensive effort in being able to rapidly make adjustments in order to steer the organisation in the right direction [10].

What typically defines a startup is the highly uncertain environment it operates in, the lack of resources, a small team and a gravitation around one, perhaps a few, products. Clear structures and rigid processes are normally absent - demanding the organisation to heavily rely on the empowerment of each team member (ibid.).

4.1.1 Lean Start-up

In an attempt to address the above mentioned issues - and to ease the burden of the entrepreneurs - the Stanford ex-student Eric Ries together with his professor Steve Blank created a new methodology called the lean start-up. The approach has not only received increased attention in the startup communities but is also becoming more and more recognised by various scholars and institutions. Prestigious universities, such as Stanford, are currently including the approach in their curriculum - further indicating the importance of being able to provide a general framework on how to establish and develop a startup [2].

The lean start-up is tailored to provide a smoother approach by favouring experimentation over the uncertain resource-heavy and rigid traditional ways. Instead of trying to presage the market and predicting what it might want, a minimum viable product (MVP) is constructed to be able to almost instantly assess customer feedback. The idea is to start a cycle in which the MVP is iterated in and each time adjusted accordingly with the information received from the potential future buyers (ibid.).

In the case of a deadlock, meaning that the iterations of the MVP are not leading to desirable market responses, it is necessary to perform a pivot – a fundamental redesign of the entire plan in order to test a new hypothesis (ibid.).

4.2 Open Source

An open source (OS) project is defined as “any group of people developing software and providing their results to the public under an Open Source license”. This is an increasingly common phenomenon in the software world from which many successful products have been derived [3].

4.2.1 OS as Part of a Business Model

There are several ways for a startup or a company to take advantage of OS. As an example, a plethora of OS software exist, allowing to avoid unnecessary spending on expensive licenses. This thesis however, will focus on OS community - how to build and maintain it as well as attract developers, specifically with help of RE. According to various studies, these communities appear to have an upper hand when it comes to innovation [27]. Moreover, OS development seems to grant an edge in the ability to more rapidly respond to bugs [4].

More specifically, as Chesbrough and Appleyard – researchers from UC Berkley and Portland State University – define it, four major models will be investigated: *Proprietary Extensions*, *Dual License*, *Device* and *Community Source*. The first two models are categorised as a hybridisation, meaning they are a combination of open source and proprietary versions. The third and fourth model falls under the category Complements (vendor sells hardware containing open source, such as Android) and Self-Service (community where applications are jointly developed to be used by all), respectively. Apart from the Hybridisation, Self-service and Complements categories a fourth category titled deployment was found which includes models where revenue streams are based on support and consulting services targeting open source software. As the authors also mention these categories are not mutually exclusive, hence, it could be that one startup turns out to pursue not only some but perhaps even all of them, simultaneously [5]. Furthermore, Henkel from the Technical University of Munich uses the term *Selective Revealing* as he revealed that many companies only share parts of their source code while keeping the remaining part proprietary [11].

Due to its many successes, businesses are now attempting to leverage on the OS approach. One project, by many recognised as very successful, is called OpenStack and involves more than 200 firms as well as many individual contributors. Interestingly, there even exist competition among the collaborators for the same revenue model. This shows how beneficial OS projects can be when businesses are willing to cooperate despite the risk of potentially losing market share to rivals [26].

4.2.2 Open Source Licenses

There is a plethora of ‘standard’ open source licenses available to choose from[13]. One could theoretically create a new license specific for the project at hand but the ‘standardised’ licenses exist to effectively communicate to a potential contributor what their rights and obligations are. In March 2015 the most popular licenses on GitHub were MIT, Apache and GPLv2/v3 and were collectively used in more than 3/4 of all published licensed projects[12]. There are some subtle differences between the different licenses but the major differences between them concern how derived work and patent usage are affected.

Sharing work Open source licenses can be categorised as either copyleft or permissive. Permissive licenses, such as MIT and Apache 2.0 will allow any type of use of the software, including reselling it as is and using it as an integral part of the product. Permissive licenses are sometimes called 'academic' licences. Copyleft licenses, also known as viral licenses, are not as lenient as permissive licenses and require that any modified work be released under the same license, or in some cases a compatible license with a copyleft clause. Copyleft licenses can further be categorised as either weak or strong copyleft. Weak copyleft licenses, such as LGPL, only requires that work done on the open source component be published with a copyleft license. If, for example, an open source library is published with a weak copyleft license that does not require the derived product to be published as open source. If, however, a strong copyleft license is used, such as GPL, any work based on the open source component has to be published with a strong copyleft license as well. This means that if a company tries to build a product that utilises a strong copyleft license, the entire product could potentially be required to be released as open source[13].

Patents If a company owns patents, open source licenses will typically grant contributors and users free use of them. Patent usage is separate from copyleft and there are weak and strong copyleft licenses with patent clauses as well as permissive licenses available. Of the most popular licenses listed above MIT is the only one without a patent clause[13].

Hardware licenses Open source licences for hardware blueprints are typically not released under the previously mentioned licenses. The Creative Commons Attribution 4.0 International license is a popular open source hardware license. In its most basic form the license simply requires attribution to the original creator and any modified work and commercial use is allowed. This basic licence can then be complemented with clauses regulating adaptations of the original work and commercial use. Adaptations can either be allowed (similar to permissive), explicitly forbidden or subject to a 'share alike' clause (similar to copyleft). Commercial use can then be either allowed or forbidden through another clause[6].

4.2.3 Open Source Community

Importance The importance of user participation in open source communities have been confirmed by many studies [18]. In some cases, the number of active developers (counted as members doing at least bug reporting) can even be used as a measurement of success of an open source project [7].

Roles An open source community can be described with the commonly used onion model [17]. As can be seen in figure 2, it contains eight different types of contributors, with the project leader at the core and each subsequent layer representing a different role with decreasing degree of influence, ending with the outer layer of the passive users.

The passive users are 'non contributing' in the sense that they only use the system. The readers not only use the system, they also read the source code. The bug reporters and fixers find and resolve bugs. Peripheral users contribute

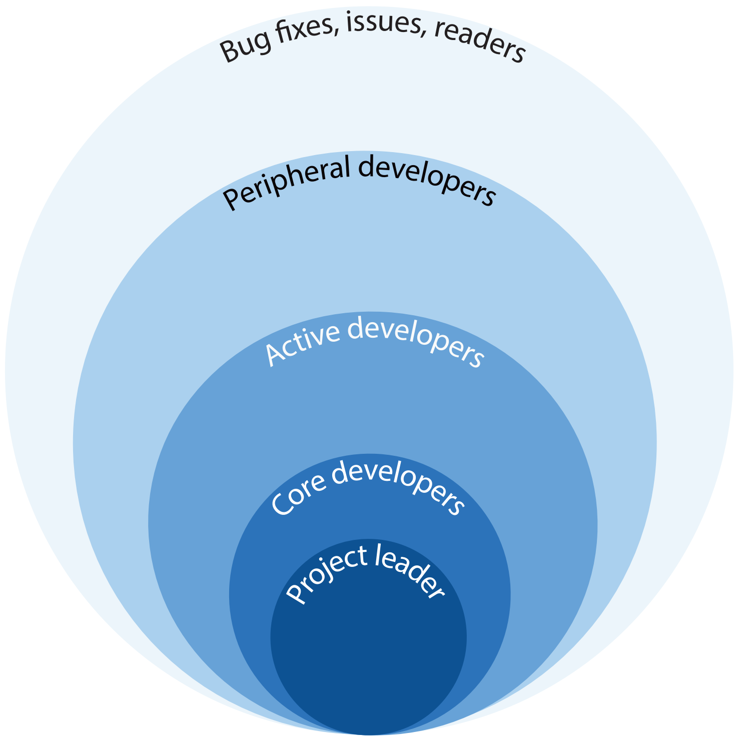


Figure 2: Onion model

new features occasionally while active developers do so regularly. Core members take on a coordinating responsibility and have been involved in the project for a longer time. The project leader is the person responsible for the direction and vision of the project.

Types of OS communities Research on open source communities have traditionally been done on so called 'community founded' communities. This is the classical type of open source community that starts with a programmer trying to scratch a personal itch. If other developers find the project interesting or valuable to their personal needs, they will join in and the community will grow organically. Linux is the prime example of a successful community founded project.

However, many hugely successful open source projects such as Mozilla Firefox, MySQL, Darwin, OpenOffice and Eclipse are sponsored projects, i.e they have emerged from a firm. Whether the software was released as open source after a proprietary attempt or was designed from the beginning with open source in mind, the governance models are different. It is in the interest of the firm to steer the project in a direction that adds to the firm's bottom line, and these goals might not align with the goals of the community. This tension between control and growth[29] is something that firms will need to learn how to manage in sponsored communities.

Depending on how a startup chooses to manage a particular project and when it is released, the type of community can essentially fall under either category

License and legalities There is a plethora of licences available to choose from with different characteristics that governs how the derived work can be used. Permissive licenses have few restrictions and will generally allow derived work to be closed as proprietary software and commercialised while copyleft (also known as viral) licences require that derived work be put under the same license[13].

Depending on the intended use from a potential contributor or user, this could be a deciding factor in whether she wants to use the project or not. If one license is not suitable for the entire project, multiple licenses could be used[14].

Initial state of software There has to be some software to base the project on. Trying to build a community on a set of ideas with no 'skeleton' to build on is most likely not going to work. Software quality is of course important but as far as quality of the initial code goes, it is not crucial. The code should be functional and as Raymond puts it: 'present a plausible promise', i.e show the observer that the project has potential to become something usable [22]. Future developers can always help with bug fixes and while poor documentation might be a threshold for some it is not imperative for success. It should of course improve as the project evolves.

Software architecture Having a well thought out, extensible and modular architecture makes it easier for developers to understand design decisions and contribute to the code base. A good architecture will help communicate to the developer how the software is put together and where the intended contribution should be placed. Designing the architecture to be inherently extensible and modular will remove some of the friction related to ease of contribution, especially if it means that a developer only needs to understand a subset of the code and not the entire system. If applicable, a plug-in system may be designed to reduce this friction even further. In this case the design choice may allow for contributions and let the main software remain proprietary.

4.3 RE in Startups and Lean Start-up

Due to the nature of startups, the RE mostly resembles an agile process. Also, a recent mapping study disclosed that the research data available is insufficient in order to fully understand the entire process [15]. The lean start-up theory will therefore be used as a complement and foundation to further describe the RE practices in a startup.

4.3.1 Requirement Specification

Specifications are generally not only ill-defined but from a RE point of view startups often lack these in written form as well [21]. The reason can be attributed to the avoidance of overhead and processes the organisations tend to have in order to remain flexible and agile. The general agile approach specifies requirements in a so called user-story form, which define a set of functions that the customer wants.

4.3.2 Elicitation

The research clearly emphasises the importance of customer involvement and as a matter of fact, a major trend in startup communities today is to elicit by literally testing the way to a final product [10]. From a lean start-up perspective, the MVP is used as a tool to elicit requirements (see figure 3). This highly agile method is diametric to the traditional product development not only in the sense of its rapid and repeated cycles but also due to the openness of the entire process. Before the revolution towards transparency, stealth-mode was favoured by the belief that someone or some organisation might steal the business idea.

There are two major benefits of lean start-up:

1. It lowers the cost of getting the first customer and reduces the risk of creating wrong products.
2. Short development cycles.

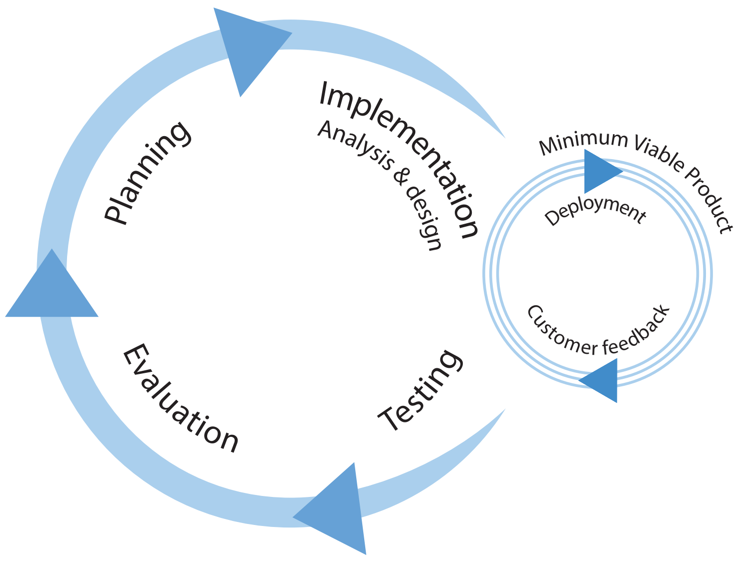


Figure 3: MVP - iteration cycle

4.3.3 Release Planning and Validation

Fast releases shorten the lead time from idea to production and several reports show the benefit of frequent deployments [21]. The lean start-up's MVP, as seen above, favours this technique. The validation tests are most often performed in connection with the elicitation on the market.

4.4 RE in OS community

Requirements engineering in open source communities is vastly different from classical requirements engineering. Classical requirements engineering offers a plethora of tools to methodically elicit, specify, analyse and validate requirements of a project. In open source projects, the requirements engineering process is done much more informally and classical requirements engineering practices are rare and not very pronounced [1] [24].

4.4.1 Elicitation

Requirements elicitation is an area where OS projects differ substantially from traditional software projects. Instead of a formal elicitation technique with interviews, focus groups, workshops and other traditional tools; requirements are often acquired informally through user input or from developers in forums and issue trackers [24] [9] [18] [20].

Developers and other community users are a major source of requirements for open source projects and are very important for open source projects [18] [20] [8]. In a corporate sponsored closed sourced project with an intended customer there is a considerable effort associated with the elicitation process. Communication needs to be precise, especially if developers do not have domain knowledge of the problem, and is prone to many errors on several different levels. The customer might not even know what she needs but somehow the intention needs to be decoded to specific requirements that the developers can program.

This problem is largely eliminated in open source projects. Since developers are users themselves[18] [22], requirements may be asserted rather than elicited; i.e a developer realises she needs a specific function, implements it and if it is merged into the project it becomes a requirement [24]. Rather than 'pulling' requirements from the customer, requirements are 'pushed' from the community.

4.4.2 Requirement Specification

In a project where classical requirements engineering is applied it is common to find a document listing all the requirements of the product, a system requirements specification document. Requirements are normally described on several different domain levels, as both functional and non-functional requirements, and are supposed to communicate to developers what the final system should be capable of. It can also act as a contract between a customer and a software developer.

In open source projects this method of doing requirements specification is very rare. Rather than having a centralised repository of requirements in a single document or a database, open source projects tend to have very loosely defined requirements scattered across forums, e-mails and other communication channels.

While classical requirements are intentionally very specific, requirements in open source projects are often vague and relies on knowledge of the project and context. They are often described in relation to a previous version or a competing product [1].

4.4.3 Release Planning and Validation

Feature-based release strategies are common for proprietary software but for open source software; time-based releases have many advantages. With a feature based strategy, developers sometimes try to rush their features into a release because they do not know when the next one will happen. A predictable schedule makes it easier for developers to plan and if combined with frequent releases, the penalty of not adding a feature to the next release is not so grave. Sharing code more often also alleviates some of the issues caused by a geographically dispersed workplace by keeping the difference to the latest software version small [16].

One of the 19 principles in Eric Raymond's popular *'The Cathedral and the bazaar'* is *'Release early. Release often. And listen you your customers.'*. Releasing early and often means early feedback and validation from users. It also enforces the view of users as co-workers or developers[22].

Since developers often are users of the software themselves, and elicitation is heavily influenced by developers, 'requirements' are validated implicitly through discussions on forums. A formal validation process is rare in an open source project [18].

4.4.4 What to Reveal

There seems to be few limitations on what types of software one can release as open source. Successful projects have been observed for consumer applications such as Firefox, large and complex operating systems such as Linux, infrastructure as the Apache Web Server and highly specialised software such as OpenEMR[19].

As for the qualities of good open source software, a study done by Stürmer and Myrarch on open source community building found some positive preconditions that might increase the chances of success for a project. Some of the factors identified were public demand, great initial source code and a degree of novelty [25].

As with any other product, there should be a public demand to motivate developers to contribute to it. Although public demand is difficult to predict, Eric Raymond offers some insight in *'The Cathedral and the bazaar'* where the first principle states that *'Every good work of software starts by scratching a developer's itch'* and the 18th adds that *'To solve an interesting problem, start by finding a problem that is interesting to you.'*[22].

Although Stürmer and Myrarch argues that the initial state of the source code needs to be *great* they seem to agree with Raymond who instead argues that the software can be buggy and incomplete as long as it runs and developers can test it. The important point is to convince prospective contributors that there is *potential* in the software. Stürmer and Myrarch also suggest that a modular architecture and documentation are important, but not vital at the inception of the project.

The degree of novelty is not necessarily defined as something entirely new and innovative. The novelty can also present itself as new features, improvements over a competing project or as an open source alternative to a proprietary solution.

5 The Startups

This chapter presents the interviewed startups as well as provides descriptions of their business models and experiences.

5.1 Mapillary

Mapillary was founded in 2013 in Malmö, Sweden. Their main idea is to provide geographical area overviews on the internet, in the same way Google's street view does. However, Mapillary focuses on providing not only street view but other places inaccessible to motor vehicles as well. The startup does this by building a strong community through which they receive a lot of data, that in turn gets published on their platform available for everyone.

According to their website, Mapillary currently has around 57 thousands photos covering an area of almost 1,4 million km.

One of Mapillary's major investors is the California based Sequoia Capital, an American venture capital firm, well-known and respected for several financial injections and involvements in companies such as Apple, Google, Oracle, PayPal, Stripe, YouTube, Instagram, Yahoo! and WhatsApp[2], marking the hallmark of Mapillary's entrepreneurs and the business model itself.

The interviewee is one of the founders, Peter Neubauer – an entrepreneur with vast experience in startups and open source. To put it in his own words:

> *I have been chief of software development in a couple of firms and I am one of the founders of Neo technology. I resigned from Neo and founded Mapillary because it became too big for my taste (approx. 130 employees). We started out with three persons 2007.*

5.1.1 Business Model

The underlying technology that Mapillary created somewhat resembles systems such as Git, enabling them to capture versioned photos in motion coupled with highly complex automated data extraction. To exemplify, this means that a street could be photographed in June and once again in July to later be analysed with help of Mapillary's computer vision, allowing to determine whether some specific objects – such as temporary checkpoints, construction work, temporary signs, etc – are missing or not. The technology is not limited to streets or bound to photos by a single vendor, meaning that if a single picture is taken of a monument from one angle and another one is added from a different perspective, Mapillary will be able to make sense of them and stitch them together. The more photos Mapillary obtains from its users, the better the result will be. The technology essentially allows for a versioned mapping of the world.

Mapillary has a hybrid approach, a proprietary extension, where the community side of their project allows for massive data collection through crowd sourcing and a professional side as well, where companies can subscribe to the underlying technology in various degrees of involvement. To incentivise the community Mapillary is free for personal, non-profit and educational use. In addition, some core components essential for the viewing experience are open sourced. Mapillary offers four different monthly subscriptions for professional

[2]`https://en.wikipedia.org/wiki/Sequoia_Capital`

use: 'professional', 'team', 'organisation' and 'enterprise', with various levels of access to the technology and content. The lowest tier subscription, 'professional', simply gives access to Mapillary photos. Every following tier builds on the previous one where the 'team' subscription allows for web map viewing and editing with the Mapillary photos and the 'organisation' subscription gives access to automated data extraction. The final 'enterprise' tier is negotiated on a case- by case basis and includes a license to repackage, reuse and resell the images and data as well as support for private photos and custom solutions offered by Mapillary.

Mapillary already has some customers – mainly cities, municipalities and map makers. Currently the startup is heavily investing in the computer vision, machine learning and AI aspects of the software.

5.1.2 OSS Strategy

Mapillary's strategy for open source is a type of selective revealing, that involves the release of several independent projects that are needed for their product to function. In cases where the startup has found a solution to a problem it anticipates having other applications, it will simply evaluate whether the overhead and general interest is worth the cost. So far Mapillary has three major projects open sourced:

OpenSFM is a python library for 'structure from motion' and is built on top of another open source project, OpenCV. SFM is a technique used to produce 3D estimates of a series of 2D pictures in 'motion'.

Traffico is a font made for traffic signs. Mapillary built this to visualise traffic signs in a vector form after their image recognition software had identified traffic signs.

MapillaryJS is a Javascript viewer for Mapillary's street level photos that was developed after they realised that the previous solution, built with ThreeJS and AngularJS, had too many dependencies to allow for easy integration with other map libraries. The new viewer easily integrates with libraries such as Leaflet, Mapbox and Google Maps.

These packages are all integral parts of the final product, but have been separated and generalised enough in order for developers outside Mapillary to find other use cases as well. It is particularly important to note that these projects combined do not make up the entire Mapillary product. The startup picks open source candidates in a very conscious and strategic manner.

OpenSFM was originally open sourced partly due to some proprietary algorithms being used. Neubauer says that it is *'protection against patents'* – making it a public domain precludes the possibility for someone else to be granted a patent. Furthermore, he emphasises that they *'are opposed to patents'*. Apart from patent protection, OpenSFM also resulted in the hiring of an employee who was an active contributor to the project.

Traffico was open sourced specifically to help with content creation for the fonts. The team originally had support for 50 different traffic signs which was far from what they required to cover the world's traffic signs. At the time of writing, content to 27 countries has been contributed.

5.2 Neo Technology

Neo Technology was founded in 2007, in Malmö, Sweden. The company's flag product is called Neo, a graph database used by many organisations such as Walmart, Cisco, eBay, Hp, and Lufthansa Systems[3]. The graph database allows for complex relationships between dependencies to be analysed in a thorough manner, not achievable by regular relational databases such as MySQL.

In the previous section, the description of Mapillary, it can be seen from the quote that Peter Neubauer also participated in founding Neo Technology. Hence, he is the interviewee for this startup as well.

5.2.1 Business Model

Neo offers two different versions of Neo, the community and enterprise editions – a dual license approach. The community edition is a standard OS project with a GPLv3 license that is available to the public for internal organisational or private use. The enterprise edition is offered with four different licenses and adds functionality related to performance and scalability compared to the community edition. The various enterprise licenses are available as a purely commercial license, an evaluation license that allows for a trial period in commercial use, an educational license and a special open source license. The latter mentioned license gives access to the enterprise edition under an AGPL license.

5.2.2 OSS Strategy

Neo was originally developed as a proprietary solution to solve problems within content management systems. The original SQL database solution was unable to handle the complexity, forcing Neubauer to invent an entirely new graph based database. Despite the new database actually being first put into production in 2002, even well before the NoSQL movement started, Neo wasn't founded until 2007. Neubauer explains that the system was ready to be open sourced in 2003, but it was deemed to risky due to the economical climate. After putting the project on hold for five years no competing product had yet to enter the market and a decision to launch was made. They soon realised that *'we can't support a full blown database in a company with 10 employees'* and that what Neo was building is *'a component that is really good infrastructure'* – making it an obvious decision to release it as OS. The original intent of the OS decision was to ensure the database's high quality without having to hire a large group of developers. A decision to release an enterprise edition was later made which required a more restrictive license. Because the code included external contributions this required permission from around 50 developers. The developers had all signed a Contributor License Agreement (CLA) prior to having their code included in the Neo code base.

Since the year of Neo's foundation, the startup has experienced an outstanding growth, putting the early days of the startup behind. Also, Neubauer explains that the initial goal of the OS community was to build a developer community but as the company and the product matured the focus has since shifted to creating a user community instead. A large part of the reason was the *'enterprise customers who really dislike if things get unstable'*. The effect

[3]`http://Neo.com/customers`

of this is that most of the development is conducted internally by employees at Neo while external contributions are really difficult to include. Neubauer further notes that the transparency provided by the OS aspect is still profoundly important, especially in alleviating the bug fixing process. This basically means that community members could take part in the bug investigation; however, not necessarily have any success in being able to perform a commit.

5.3 Bitcraze

Bitcraze was founded in 2011, in Malmö, Sweden. They develop and manufacture a small quadcopter called the Crazyflie. The software is open source, allowing customers to make compelling adjustments in order to satisfy their true needs. Because of this, Bitcraze has managed to reach out to scientists and scholars from places such as NASA, Stanford, Microsoft, MIT and ETH Zurich[4].

The interviewee's name is Kristoffer Richardsson, a developer with vast amount of experience within IT and agile development. He has both worked as a consultant and an employee with everything from being an IT architect in embedded systems to agile coaching.

5.3.1 Business Model

Bitcraze are not targeting a broader audience and hence are not having an active focus on expansion, except for one reason, namely to *'be able to do more cool stuff'*. Moreover, the startup wants to focus on creating *'a flying open development platform'* for educational use, creators and researchers, instead of focusing on profits. This is also reflected in Bitcraze's reluctance to bring in external investors.

Everything ranging from hardware blueprints to the source code is open sourced. In order to maximise the time available to support itself and the community Bitcraze are outsourcing the manufacturing and distribution of the final product to a company called Seeed studio. The license restrictions that allow Bitcraze to generate revenue from this model are the *share alike*, *attribute* and *non commercial* clauses. More specifically, it encourages the purchasing, production and modification of the hardware as long as it is shared publicly, attributed Bitcraze and not used for commercial purposes. As the license pertains to the hardware blueprints there are no restrictions on what a company can do with the hardware if they buy it from Bitcraze. The startup is earning money by selling their hardware, a business model that would fall under the complement category.

5.3.2 OSS Strategy

Bitcraze's open source strategy is fundamentally different from the other startups involved in this study. As Bitcraze's business model is essentially to produce *'a flying open development platform'* everything from the software to the hardware blueprints are made open source – in other words, everything is open source. The strategy is largely grounded in an ideological view, as Bitcraze's Richardsson puts it, *'values are the biggest motivating factor'* for open sourcing. Bitcraze even open sourced their website *'because, why not?'*. A large part of the rationale concerning this attitude is explained by Richardsson as *'if you're open, you're open for help, if you're closed you can't get any help'* and that if you still do no get any help *'there was no difference, but if you do get help it's better, so why not? What's the point of closing stuff?'*.

Although Bitcraze had considerable amount of initial interest when the product was *'early adopter hot'* Richardsson discloses that they have since lost a sig-

[4]`https://www.bitcraze.io/`

nificant part of it, mainly due to the mismanaged communication with the OS community. This rise and decline of contributors is internally referred to as the first phase – whereas the second one is the contemporary attempt to recapture some of the community activity that has been lost. The approach consists of having an employee spending more time managing the community and working on raising transparency of their internal development process, in particular by communicating long and short term goals more extensively.

5.4 RefinedWiki

RefinedWiki (also abbreviated Refined) were founded by interviewee Emil Sjödin and his co-founder in 2009. The startup currently employs ten people and their main product is RefinedTheme, an addon to Atlassian's Confluence software. The component allows a company to tailor their Confluence experience according to the needs, by managing layout and organisation as well as styling for the brand.

Sjödin was studying Information and Communication Engineering Technologies at LTH when UI company The Astonishing Tribe (TAT) hired him to produce an intranet using Atlassian Confluence. Realising that the platform could provide more business opportunities he decided to start RefinedWiki instead of finishing his master's thesis.

5.4.1 Business Model

Refined are heavily dependent on Atlassian's ecosystem and are exclusively producing addons to their Confluence and JIRA platforms. They have no plans on creating their own intranet solutions or standalone products and would rather focus on addons for other platforms should something happen with Atlassian or their partnership.

Refined are focused on the engineering aspects of the product development and are therefore outsourcing sales to consulting partners. The partners in turn are the ones who not only sell but communicate with the end customers as well. Refined also sells their products through a marketplace provided by Atlassian.

Apart from their main product, RefinedTheme, Refined also provide a mobile experience for Confluence, a ToDo plugin for Atlassian JIRA and a free Confluence addon with standard UI components.

The theme, the mobile experience and the ToDo plugin are offered with the same subscription model applied by Atlassian that is based on the number of users. There is also a 50% discount available for academic institutions and a free license for open source projects, non-profits and classrooms. The qualification for a discount or free license is handled by Atlassian.

5.4.2 OSS Strategy

Refined just recently released their first OS project and have currently no formal strategy for their open source ventures. The approach could be seen as a complementary asset and proprietary extension. The project is a tool used to alleviate some of the issues they faced while developing products that should integrate with two different Atlassian addon environments. The difficulties stemmed from that the previous environment they developed for was, in comparison to the new one, far more open. The new tool Refined developed allowed them to easily integrate the same addon with both environments. The developer responsible for this asked if he could open source it. Sjödin then decided that the advantages of *'showing commitment to Atlassian'* outweighed the possible disadvantage of helping their competitors. Sjödin further states that they have no real hope of getting contributions; branding was the main motivating factor and *'it's more fun for the developers to be able to show what they have done instead of keeping everything internally.*

Table 1: Summary of Business Models - Source: [5]

Category	Model	Description	Startup
Hybridisation	Proprietary Extensions	Firms broadly proliferate open source application and monetise through sale of proprietary versions or product line extensions.Variants include mixed open source/proprietary technologies or services with free trial or "community" versions.	RefinedWiki, Mapillary
	Dual License	Vendor licenses software under different licenses (free "Public" or "Community" license vs. paid "Commercial" license) based on customer intent to redistribute.	Neo Technology
Complements	Device	Vendor sells and supports hardware device or appliance incorporating open source software.	Bitcraze

5.5 License and Legalities

Releasing copyrighted assets or intellectual property without proper consent could be a serious problem and if the source code utilises other open source projects, the license agreement of that project needs to be honoured. As an antithesis to these potential legal issues, releasing software as open source can also be a strategy to mitigate legal issues as explained by Mapillary:

> *We released a module that converts 2D pictures to 3D. This was released due to the fact that we didn't want to own a lot of proprietary algorithms. We also looked at it as a protection against patents. We are against patents.*

The interviewed startups had a variety of different licensing models. In the case of Mapillary they released all software under permissive licenses because it allowed easy contributions and open source is not the main reason why the company exists.

> *In Mapillary everything is released under MIT, BSD, Apache and the likes. We're not interested in making money or forcing people to contribute. Contribute with no strings attached, it's only helpful.*

Neubauer also notes the difference when compared to Neo. Neo uses copyleft type licensing, meaning that derived work needs to be published under the same license. The reason in this case being that Neo are not primarily looking for contributions due to the complexity of the software and nervous enterprise customers. Instead they are trying to amass a 'user community' where participation is largely in the form of bug reporting.

Bitcraze on the other hand is entirely open source based and have a variety of different licenses, permissive and copyleft, for various reasons. They note a potential problem with this.

> *We can only use the code we ourselves actually wrote. We could close that and for example sell to another company as a product. But if you committed code to our code tree then that code cannot be there.*

Because Bitcraze develops an open source quadcopter, the blueprints for the hardware are available as open source. They are released under a Creative Commons license with share alike, attribute and non-commercial clauses, meaning that derived work needs to be available under a similar license, be attributed to Bitcraze and cannot be used for commercial purposes. This means that only Bitcraze are allowed to mass produce and sell the hardware. Unfortunately this has not stopped immoral manufacturers from producing counterfeits.

> *They [the immoral company] took our blueprints, produced their own and are selling them.*

Regarding the risk of someone taking the software to develop a competing product neither Neubauer nor Richardsson were particularly worried. Bitcraze noted that forking without contributing back to the original source was a bigger issue than competition.

All of the startups had at least one employee who had previous knowledge of open source licenses and none of them thought that it was particularly hard to deal with.

6 The Internal and External RE Process

Once the OS community is established the question is how to make sure it remains prosperous and alive. At the same time it is important to ensure the startup's consistency and capability in being able to provide support and help. Moreover, and perhaps most importantly: how can the seeds from an OS community be harvested?

6.1 Elicitation

The process of eliciting requirements are in this section categorised as either internal, i.e from within the startup, or external, i.e from partners or the open source community. The results on how the startups managed these activities are presented.

Internal Of the four startups studied, the emphasis on internal elicitation was greater in Refined and Bitcraze than in Neo and Mapillary. The two former startups had both implemented a formal elicitation process popularised by Google, the 20% method. The approach dictates that 20% of working hours should be spent on a personal project. Bitcraze calls the method 'Fun Fridays' and Richardsson explains that every Friday is spent on a side project. He says that *'the idea is to play around with things that we're not sure are going to work or where the idea will end up'* and *'as long as you're doing something that you think is fun you can do it'*. If anything done during the 20% time is deemed valuable the project is elevated and a product is formalised around it.

Both startups successfully created complimentary products through this technique. Bitcraze developed a local positioning system for the Crazyflie 2.0 designed to give an absolute position in 3D space in an indoor environment. The system was released as an expansion deck and will greatly enhance the capacity of autonomous flight. In Refined's case the 20% method spawned RefinedTodo, used internally for task management and later released as a plugin to the Atlassian platform.

Apart from the '20%' method, no other formal elicitation techniques were observed. Part of the reason why was explained by Richardsson saying that *'ideas is not what's lacking, we have a thousand billion ideas that we want to implement. The problem is to decide every week what we think is most important at the moment.'*. Informally however, both startups leveraged the fact that they are small companies and encouraged employees to speak freely when they had an idea. Sjödin credits the lack of formal elicitation techniques to the size of the company and says that *'we're 10 employees, so we're not as governed by a specific process as bigger companies are'* and *'if anyone has an idea they could just shout it out.'*

Neo and Mapillary have brainstorming sessions where everybody can contribute their ideas.

External The practice of eliciting requirements from external sources, such as partners and the open source community, was prevalent at all startups. Although no formal techniques were observed, all startups had multiple communication channels that allowed for external elicitation.

Established partnerships were heavily used at Refined as they were effectively decoupled from their end consumers through partners selling and customising their products. This also meant that they very rarely elicited requirements directly from the costumers that actually used the products.

Contrary to how Refined operates, Neo and Mapillary actively utilises their open source communities to generate customers and elicit requirements from those customers. If they receive the same request from several sources the process of implementing it to the startup's roadmap will escalate. Neubauer explains that following the accelerating growth of Neo and the subsequent increased difficulty for external participants to contribute code to the repository, the community has adopted a different role. He notes that *'open source is the main component for lead generation and community building in Neo'*, adding that while they currently employ sales personnel, that was not the case in the early days of the startup. During the first years Neo relied exclusively on the open source community to generate customers and Neubauer expands on the advantage with the approach, saying that *'people come to us and tell us what they want to do and what they can already do'*. He emphasises the fact that the potential customers already have an intimate knowledge of the product through the open source project. Having tested the system and analysed how they can use it Neubauer says that the customers will *tell us what they want to pay for and how the product should be* but more importantly *'how they can help us get there'*

The situation is similar for Bitcraze although they have not had the same success with requirements elicitation. As mentioned by Richardsson earlier, the problem for Bitcraze is not in the process of generating ideas but rather the lack of time to implement them. There are channels set up for discussions about what the quadcopter is potentially capable of, notably the Bitcraze forum, and it does exhibit activity but less than during the infancy of the startup. Richardsson says that *'people solve a lot of problems and do a lot of interesting things, but they don't really share it'* explaining that while there might be a lot of features for the product, the startup has a problem with curating them and making them official Bitcraze features.

6.2 Requirements Specification

Requirements Specifications (RS) differ not only due to cultural characteristics but also on the type of OS-strategy the startup pursues. Furthermore, one has to distinguish between internal and external RS; more specifically, the written communication in the internal organisation versus the one in the OS community as well as including the bridged RS between these two.

This section will be divided into the aforementioned categories; internal and external/bridged.

Internal The internal RS described here is, if nothing else is mentioned, not accessible to the OS community.

Generally, the startups don't invest much time in writing down RS; however, Mapillary's Peter realises that *'It could've been better and I would like people to write more in order to make it more lenient for others to take over when needed; for example, if someone goes on vacation. We are only 1-2 persons per component'*.

Moreover, RefinedWiki explained that their RS activities evolved once the company grew and new departments took shape, quoting Sjödin, *'We actually started out with requirements in connection to hiring a new employee for testing tasks. Suddenly it became natural to have concrete requirements specification'*.

In order to not spend excessive amount of time in specifying requirements, *'The level of abstraction is very high'*. Mapillary's Peter mentions how they usually map the data architecture's design while at the same time taking notes of important aspects they may have to consider. In Mapillary's case, the design requirements are written down in Dropbox paper or Google docs to later be broken down into tasks, created in Trello. Similarly, requirements were encoded in tasks and stories in Trello at Neo.

Similarly, RefinedWiki's requirements are mostly kept on design level; however, they also specify *'how the functions will work in general in order to be able to create somewhat reasonable test cases'*. Sjödin states that both requirements combined provide a good overview of how the product actually will work.

In contrast to Mapillary, RefinedWiki uses their own system (wiki) to write down requirements. They use their own developed ToDo application as well, mostly for keeping track of ideas.

Bitcraze uses the old fashion post-its on a board solution to write down tasks for the coming week. However, as highlighted by Kristoffer;

> *If there is something of great importance we'll of course save it. Most of it is located in our heads as well as in our long-term direction [vision], you could say. This is pretty much how we document things.*

Kristoffer also states that he doesn't believe it's necessary to write down ideas, as he mentions that *'I personally think that if you have an idea, it will come back to you. You don't have to be afraid that it might disappear. It probably wasn't a good idea if you forgot about it'*, reflecting Bitcraze's organisational culture. On the other hand they realise that even though the requirements could end up as waste, some sort of plan or direction is still needed. Kristoffer eventually says, *'It's a difficult balancing act but we are committed to minimise as much as possible'*.

External As mentioned in the introduction the RS procedure depends on the type of OS approach the startup has. Mapillary does this by splitting their work into distinct components, allowing them to outsource work as separate entities decoupled from the main projects. Interestingly, this enables a unique way in specifying and communicating requirements, namely, by providing a clear vision as a fundamental part of the component to be outsourced. This, according to Peter, is achieved by creating something *'that makes it simple to start and that is already working and displays the vision'*. Although Peter consistently remarks that excellent code quality and documentation is imperative in order to build an OS community successfully, the component itself shouldn't be flawless. Peter mentions that *'it's actually good with crappy code. People feel that they can refactor here, it's going to be great'*.

When it comes to the documentation of the process regarding the OS components, such as issues and backlogs, everything regarding the open source projects is available on Github including what Mapillary is working on internally. To quote Peter, *'all issues, bugs and improvement propositions are posted in Github's issues, available to everyone. So, we even perform pull requests'*. Neo adopted a similar approach although they did not have clearly separated components. Specifications that were developed and refined internally were not always implemented by Neo but outsourced to the community through issues.

In contrast to Mapillary, Bitcraze's OS approach differs in the way that they strive to be as much open source as possible – regardless of the source code's importance to the startup. Subsequently, communication and requirements turns out to be crucial and more importantly intertwined with each other. Bitcraze defines their OS community development with two phases; the first phase described as when Crazyflie was state-of-the-art, hence, a time with many early adopters available – and the second as being the contemporary one. Moreover, Bitcraze's Kristoffer points out that they mismanaged the first phase by not providing sufficient support to the community – which apart from Crazyflie becoming somewhat outmoded – has lead not only to a massive decline in general interest but in amount of contributions as well. In addition, as opposed to Mapillary, the startup's community structure is heavily proliferated such that there exist around 30-40 different repository for Crazyflie. This makes it inherently complex, requiring even more assistance for outside developers. Kristoffer explains that *'most of the time you have to change something in multiple places'* and continues to state that *'the complexity is a reason why it would be good to have clearer issues and features so that people can contribute more easily'*.

This has been a major concern for Bitcraze, pushing the startup towards an attempt to revive the OS community. The second phase is therefore described as *'an attempt to regain interest'*. Furthermore, he specifies what the startup believes it has to improve in order to once again attract developers, quoting Kristoffer, *'we need to communicate issues on Github as well as our internal discussions'* – the latter emphasising the need of being transparent. In an attempt to alleviate the issues faced by the complexity of their software, Kristoffer says that they have *'created a meta repo that only contains issues so that we can talk about features'* but also notes that so far it has been unsuccessful. Bitcraze also uses forums and mails to communicate requirements.

RefinedWiki are still in the process of establishing an OS community; however, they already have APIs openly available to developers. In Sjödin's words:

Our open APIs are documented. They are intended to describe the technicalities as well as to provide an overview and tutorials of how to get going. (...) These kind of things are generally well documented, also including tutorials.

6.3 Validation

Without market approval a product is destined to fail and a great deal of money can be lost, making it inherently important to attain valuable feedback. Generally, Bitcraze and Refined perform most of their validation internally whereas Mapillary and Neo also take advantage of the external approach.

As previously, this section will be divided into the categories internal and external.

Internal It is particularly critical for Bitcraze since they not only ship software but hardware as well – that could add additional unwanted expenses. This, combined with the failure of what the startup defines as first phase (mentioned in earlier sections), pushed Bitcraze to implement four different phases for their new hardware releases in order to ensure early feedback. In practice this means that the first stage is to release an early prototype to potential users, citing Kristoffer, *'We build five or so prototype systems and then we announce that this product is in an alpha stage on the blog and ask if anyone is interested in trying it out. We kind of probe the market this way'*. As a small side note, it should be mentioned that the step before alpha stage is called *developer stage*, in which prototypes that the company believe would have a market interest are created. Moreover, the alpha stage consists of, quoting Kristoffer, *'what we call 'friendly users' who might buy it at cost'*. He emphasises its main purpose being to obtain feedback on the hardware, such as *'is this what we need? Do we have the right connectors, the right size? Whatever it can be'*. After the validation of the hardware is completed, the product moves on to what they refer to as *early access*, prototypes are ordered to be sold on the web shop. This phase is also dedicated to software, as highlighted by Kristoffer *'the hardware is finalised but the software is lacking'*. Often, some fundamental drivers and basic software is available though *'but without higher order functionality'*. At this stage the startup works together with the OS community in order to produce code that the market wants. The phase cycle is completed *'when we have software with a bit more functionality and quality we'll call it a product and remove the early access logo. So the four stages are 'development', 'alpha', 'early access' and 'product'*.

Although Refined's validation process is somewhat less formal, the startup still has defined tactics on how to validate its products. One of those being a beta testing phase with a selected group of customers that they established good relations with. Sjödin prides with the fact of doing *'a very good thing for this release. We implemented a feedback button on almost all admin views'*. The feedback system allowed customers to provide comments together with screenshots. It was clear to Sjödin that *'the accessibility and ease of use contributed to increased feedback during this phase'*. Lastly, Refined concludes that *'it was a great success. I think the main difference was this button and that it's so simple to use, otherwise people won't be bothered to do it'*.

Furthermore, Refined's unfortunate position – in terms of lack of customer proximity – in the supply chain creates communication challenges with their end users. Luckily, a significant portion of their sales goes through consultants, as they refer to as *partners*, alleviating the process of collecting information. Their steady contact with partners allows the startup to more comprehensively understand what the market looks for.

When requested, Mapillary release early access versions as well. Moreover, they will disclose their products to their ambassadors before the official release. Finally, Mapillary checks with community if the product requested matches their expectations.

Neo used to provide access to snapshots in their master branch; however, this has stopped since their shift towards a user community.

External Mapillary's Neubauer however, argues that OS projects are more susceptible for feedback. He exemplifies by arguing that *'you won't obtain the proliferation and the validity [without OS] that made Mongo[the popular database] worth three billion dollars'*. Moreover, he claims, that the feedback from an OS is tremendously extensive and valuable, something Mapillary take full advantage of. When asked how they validate, Neubauer explains, *'lets say someone wants a red button instead of a green one. We put up an issue named 'make button red' and if people are interested discussions will take place'*.

Bitcraze, to some extent, use their alpha and early access stage to validate externally in the same way Mapillary would do.

6.4 Release Management

All startups stressed the importance of frequent releases in some form or another. None of the startups had a strict time based strategy for their release planning but Refined differed from the others since they had a fairly strict feature based release strategy. They noted however that they were not happy with the frequency of these releases and as a consequence they often had to strip planned features from the release. Sjödin states that *'we have released a bit too infrequently so we're going to try to do that more often. We'll try to do smaller releases to increase continuity'*. Furthermore, Refined realise *'it's important to reach the market quickly and get feedback on what you're doing'*. It should be noted that this release strategy was applied to their proprietary software.

Bitcraze's releases are more frequent; however, as opposed to Refined, they release not only the binaries but the source code as well. As Bitcraze's Kristoffer mentions, *'the source code is always available if you want to download the latest version from the repository and build it yourself'* whereas *'the binaries however, that you can download and load into the client, have been released maybe once every six months or so'*. Likewise, Kristoffer is also keen to increase the rate of releases, he specifically says that *'I'd like to release every day'*.

Mapillary on the other hand *'do continuous deployment, almost. The different components are upgraded all the time'*. This means that code is pushed into production immediately. Neubauer continues to explain that *'we don't have a staging environment, we stage in production, which is a bit controversial.* He says that they achieve this by having a very forgiving development environment and that rolling back is always possible if anything breaks. When they have developed a couple of features they think are *'cool stuff'* they will notify their community members through a newsletter.

Neo implemented a feature based release strategy. Neubauer explains that it was less structured in the beginning and a typical release would stem from a thought such as *'oh, this is a big thing for the project, let's release this as 2.0'*. As the startup matured the strategy has remained feature based but with more planning and a goal of releasing quarterly.

Moreover, the startups were asked if they were worried that releasing a product as open source too early could have a strategic disadvantage in the sense that they would allow for early forking by potential competitors. Refined had yet to gain the experience to properly answer the question but the other startups did not worry at all. In fact, Mapillary's Peter expressed it in this way:

> *The real problem is to get anyone to care at all.*

7 Analysis

The first part of this section will provide a summary of each startup's incentives and strategy and OS community development. The latter part attempts a summarising overview of the commonalities found among the firms and their approaches.

7.1 The Incentives

What are the motivational factors for a startup in order to create an OS community? Four main reasons can be identified, namely; ideology, innovation and proliferation, bug fixes and human capital.

Ideology It is evident that certain beliefs affect the development of the startup's business model. Bitcraze and Mapillary both have extraordinary passion not only for OS but transparency and collaboration in general. The former mentioned went as far as even making their website OS, quoting Kristoffer *'if you're open, you're open for help, if you're closed you can't get any help'*. Moreover, he explains *'we've received a few [contributions] actually, so that's fun'* and *'you can correct spelling errors and clarify things and stuff like that. It actually worked'*. Lastly, as the startup's flagship product is focused entirely around OS it further indicates how important ideological views are.

Mapillary's founder also expressed diligent conviction in OS. As an example, Peter mentioned that *'we are against software patents. It doesn't promote innovation, it's not nice'*. The views are also reflected in the general approach of being transparent as well as favouring lean startup. As a fun fact, Peter disclosed that the founder of this business methodology, Eric Ries, is an adviser for Neo Technology.

On the contrary, Refined is much more reserved and as opposed to Mapillary, prefer stealth-mode over transparency. It seemingly formed the startup's business model, explaining why their OS strategy is the least developed compared to the other ones.

Innovation and Proliferation Both Mapillary's Peter and Kristoffer's Bitcraze exemplified numerous times during the interview how important these factors are for their products. With help of OS, the startups manage to *'get PR, it proliferates, people speak with each other, they have fun ideas and they help each other'*, as mentioned by Kristoffer. Peter stressed that *'for Neo it was because of the fact that we couldn't maintain a huge database engine. You need help. It would also require too much attention that should otherwise be spent elsewhere'*. Furthermore, he provided a concrete example of how, as mentioned under the ideology paragraph, they protect themselves against unwanted patents; *'we released a module [open source] that converts 2D pictures to 3D. This was released due to the fact that we didn't want to own a lot of proprietary algorithms. We also looked at it as a protection against patents'*.

Bug fixes Mapillary's founder greatly emphasised what he perceived as the only way to truly address complex software issues, namely by OS. In his view, *'having more eyes, if you do it well, on the software. What's a deep bug for me* *may be a shallow bug for you and vice versa. In reality, it is a huge problem.* Also, he further noticed, that peer reviews play a major role due to the group pressure which is created by the hundreds of contributors. Apparently, he explains, *'many OS project's source codes are of better quality than closed source ones, due to precisely this reason'.*

Human capital All startups underscore the general importance of having dedicated people and the ones with OS experience can attest how helpful a healthy community is in finding those. As an example, Mapillary recruited their first 30 employees through their community. In addition, Bitcraze managed to obtain help from various scholars around the world, including from prestigious universities such as Stanford and MIT.

7.2 The Startups Summarised

An overview of each startup's OSS community involvement will be provided in this section.

7.2.1 Bitcraze

The startup's vision is evidently not only affected but also formed by the individual's view on openness and transparency, which lead them to create hardware with the clear intention of providing a fully open source platform. The firm's business model (BM), *device*, makes it extremely necessary for having a healthy OS community in order for the product to develop and in turn become more attractive to the broader market. Based on the results, Bitcraze could be positioned in the center of the model. Furthermore, only the first layer, namely, the core members, are currently available in their community. The startup is investing heavily in expanding these by attracting outside developers, which, requires time and effort from the employees. Due to, as defined by themselves, the first phase, Bitcraze implemented processess in an attempt to mitigate the risk of falling into the same trap of mismanaging the community, thus losing the interest – clearly indicating their recognition of RE's important.

7.2.2 Mapillary

Mapillary chose to release several independent projects as open source by applying. While Mapillary uses the projects as important components of the final Mapillary product there is a lot of proprietary technology involved. This is effectively a *Proprietary Extensions* model within the *Hybridisation* category. The released components have their own vision and goals and are easily managed separately. It is an efficient way of mitigating risk while simultaneously reaping the benefits of open source. Since the final product is not the sum of the released parts another startup cannot simply fork the projects, combine them and create a competing product. Furthermore, smaller separate projects have the benefit of lower complexity, lower management overhead as well as clearer focus for potential contributors. Another advantage of this approach is that individual goals can be applied. In the case of *Traffico* Mapillary needed contributions to build a database of traffic signs while the *OpenSFM* project and *MapillaryJS* projects had other goals. This means that the level of involvement from Mapillary can vary accordingly among the different projects. As *OpenSFM* and *MapillaryJS* satisfied the internal needs of Mapillary at the time of release, the startup could apply a more relaxed role in the development and management of the projects. While the startup initially assumed the Project Leader role, in reference to the Onion Model, the position could easily be changed according to Mapillary's current needs and resource constraints.

7.2.3 Neo Technology

The Neo4J graph database was created by the founders of Neo to solve issues with content management systems that traditional relational databases could not. Realising that developing a full blown database while also focusing on building a business with a very limited set of employees was infeasible, the startup decided to open source Neo4J. The founders also realised that a database is a type of infrastructure and that such components are of particular interest to the open source community. As the project grew and matured the business model shifted to incorporate a dual licensing model. This involved changing the licensing and getting explicit confirmation from around 50 contributors that they were allowed to do that. The team had made their contributors sign CLAs which aided the process. After changing licenses the software is now offered in a community edition as well as an enterprise edition. The cumulative effect of a more mature software base as well as the presence of enterprise customers subsequently resulted in a much higher enforced standard for contributed code. This means that at the current state it is extremely difficult for external sources to actually contribute to the code base. Essentially the function of the community has shifted from an active developer community to what Neo calls a *User Community*. The code is still open (for the community edition) but functionally it exists as open source to allow for external scrutiny, not for contributions. The idea is to leverage the proliferation and branding effects of open source and because the software acts as infrastructure there is a perceived increased legitimacy due to external sources being able to access and review the code base.

If Neo's position in relation the the Onion model is analysed the company has never left the Project Leader role. The startup established a strong foundation in the center of the model and decided to rely on the outer layers dynamically throughout the lifetime of the project. During the time following the creation of Neo4J all layers served an important role and as the project grew, core developers were actually recruited as employees of the startup and reliance on the community gradually shifted to only include the outer layers of the model.

7.2.4 RefinedWiki

As opposed to the above-mentioned startups, Refined has a more sceptical approach towards releasing their code publicly. Moreover, they seemingly are in the beginning of their OS venture, hence making them the least experienced firm in this case study. As the startup currently have no contributions whatsoever, it remains to be seen if Refined manages to establish some interest.

Historically, Refined have operated on a proprietary basis, which combined with their ideological views, pushed them towards the same strategy Mapillary have – proprietary extensions.

7.3 What to Reveal

Chesbrough and Appleyard reported on a variety of business models used by firms creating open source communities[5]. The models can be categorised as hybridisation, complements, self-service and deployment. In this thesis the startups primarily fell under the hybridisation and complements categories. The hybridisation category includes models where proprietary software and open source are combined, while the complements category includes a single model where software is open sourced to support a hardware product. The self-service category was not observed but would concern models where organisations develop an application cooperatively. The deployment category includes models where services related to open source products are the main revenue streams but was not observed among the startups in this thesis.

Refined, Mapillary and Neo all used the hybrid approach, while Bitcraze used a complements model. Neo's model is a dual license model where a community edition is offered for free and an enterprise edition offers additional features for a license fee. Refined and Mapillary applied a proprietary extensions model where parts of the software used internally was released as open source. Bitcraze, in contrast have no proprietary software and are using a device model. Although the hardware blueprints are open sourced, production of the hardware for commercial use is disallowed by the license.

Considering the actual software and the applicability of open source, the research does not seem to indicate any discrimination of certain projects types. Software for a variety of use cases have been observed, ranging from highly specific medical tooling, such as OpenEMR, to consumer applications, infrastructure and operating systems. However, there has been research on what qualities the software should exhibit to increase chances of success[25][22]. There should be a public demand, a degree of novelty and while the software can be buggy and lack documentation it should be sufficiently developed to convince developers that there is potential.

As reported in section 5, the startups generally followed the principles suggested by the research. The tool developed at Refined solved a problem with their development process and was released because they believed other developers in the Atlassian community could benefit from it. The reasons were primarily for branding purposes and they noted specifically that there was no product value in the tool. They were much more reserved when asked about other potentially open sourcable projects. Mapillary had several open source projects and had the most defined selection process. When they developed components needed for their product and found that something could be generalised and was of value to other developers the component was extracted, made independent and released. In most cases they wanted external developer help so the selection process existed to avoid unnecessary management overhead related to open source projects. In Neo's case they actually waited until they could observe a public demand and then released their database engine. As databases are considered infrastructure there was significant interest from developers and an apparent degree of novelty since it was one of the first graph databases released. As the database matured the startup shifted to a dual licensing model. Bitcraze had no selection process in place and released everything, including their website, as open source.

Selective revealing All participating startups, except for Bitcraze, applied a selective revealing process. Bitcraze instead opted to release everything they developed. While this could be viewed as an edge case of selective revealing, we would like to treat it as a separate model and compare the two approaches. Essentially, selective revealing seems to offer more strategic choices.

Intellectual Property If selective revealing is applied, the startup has more granular control of their intellectual property. When Mapillary released their patented software as open source, they did so by choice. Bitcraze would not have the same opportunity to make that decision if everything is already open sourced.

Project prioritization Selective revealing allows a startup to prioritise between different projects and apply different levels of involvement accordingly. As Bitcraze has released 30-40 repositories that combined results in their product, they have applied their external RE process on a product level. Mapillary's independent projects have an external RE process applied on a project level. This means that they can tailor their processes depending on the current needs of that project and the startup. As an example, the point of the Traffico project was to get contributions in the form of more traffic signs and the process can be tailored for that purpose. If they have another project that is released purely as protection against patents, an external RE process is not necessary at all. Again, as Bitcraze's process is applied on a product level the separation is not as clear.

Risk of theft Bitcraze had their hardware copied by immoral manufacturers. Although the license applied to the hardware blueprints forbids such activity, it requires legal actions to enforce. Mapillary on the other hand is not exposed to the same risk since their open source projects cannot simply be combined to replicate Mapillary's product without reverse engineering the proprietary parts.

Changing licenses Neo initially released everything as open source but when they decided to produce an enterprise version with a more restrictive license they needed explicit permission from all 50 contributors to relicense the code. The Neo team fortunately had CLAs signed from the contributors but they still needed to contact all of them and get their permission. In a situation where selective revealing is applied the risk could be spread out over many projects and potentially not require explicit permission from as many contributors if one or more of the separate projects are in need of relicensing.

7.4 The Research Questions

The research questions (RQ) are revisited here for the convenience of the reader:

RQ1: How is RE currently structured and practiced in startups engaged in OSS development?

RQ2: What challenges can be identified in regards to current RE structure and practices in startups engaged in OSS development?

RQ3: How can the problems identified in RQ2 be mitigated?

Following sections attempt to answer the above-stated questions.

7.5 RQ1: Requirements Engineering

Based on the previous section it can be concluded that the amount of resources needed to build, maintain and leverage an OS community could potentially be reduced. Furthermore, the startup's decision on which BM to pursue will also affect the level of effort they will eventually need to put in. Despite this, all startups as well as the literature seem to be indicating the same trend – transparency and openness. The following paragraphs will provide an in-depth analysis of the core components of RE.

Elicitation The internal process can be exemplified by the 20 percent rule that allows employees to spend one day every week on personal projects. This was the only observed formal technique. The reasons given for the absence of such processes were that the startups already had many ideas, but no time to implement them, and that their limited size allowed them to elicit informally instead.

Although not explicitly using the term minimum viable product (MVP), the OS optimistic startups seem to have adapted this concept. Even the most OS sceptic startup, Refined, is mainly eliciting with the help of their consultants that have access to their products. Partnerships, such as Refined's, were a common source of elicitation for all startups.

Elicitation from the open source communities was set up as previously observed by various scholars [24][20], i.e, channels were set up through forums and issue trackers and the process was done informally.

The internal and external processes had a clear distinction between them where internal elicitation allowed the startups to control the direction of the project and external elicitation was used as a supplementary process. In other words, internal elicitation sets the main focus and scope of the project as a whole while the external contributes with more specific parts.

Requirement Specification Research predicted that the startups would try to minimise process management, in part due to the agile nature of them, and thus not adhere to classical requirements engineering practices[21]. This was true for all startups in this study. Requirements were found encoded in tasks using various tools, ranging from commercial tools such as Trello to internally developed ToDo applications and post it notes. The only formal specifications document observed was found at Refined. That document spawned as a result of a recently hired tester. Mapillary described requirements on a high abstraction level but expressed a desire to formalise documentation in order to diffuse knowledge within the startup to better handle cases where an employee is absent. In general, the startups seemed to utilise their limited size to forgo a formal process internally.

The external process was also in line with what the literature has previously observed[1]. No formal specifications documents were observed and requirements were informally described in issue trackers and forum discussions. Mapillary made a point of adhering to the tools available to the open source community by posting their own internal issues and discussions on Github and even performed pull requests for features they implemented. Bitcraze expressed a desire to do the same, but stated that they did not do so sufficiently at the moment and would like to improve the process.

Validation As mentioned under the elicitation paragraph – the lean startup approach was never really defined, however, the majority of the startups, if not practicing it, were leaning towards this methodology. Also, as with the elicitation, validation took place through informal channels (known as informalism by Scacchi [24]), such as forums and issue trackers. Furthermore, validation and elicitation are tightly connected causing validation to be done implicitly by elicitation from the community, allowing some *(PA EX)* to be reused in this section.

For the most part validation took place internally with each startup having a different approach. Bitcraze's process was the most formal, consisting of specific stages where the validation took place. As compared to the other startups, Bitcraze have to perform validation twice – both for hardware and software – to some extent explaining the necessity of having more rigid plans.

Generally, the main source for validation were some kind of early adopter-users. In Bitcraze case, the so called *friendly users* were allowed to buy the hardware at cost, in exchange for providing feedback to the startup. Refined implemented feedback functionalities in their products, simplifying the process for its users. The startup also received valuable response from their consulting partners.

Mapillary had a more external approach, leveraging the OS community by enabling developers to post issues and discuss them together.

Release Management Release management can be done based on features or time. The literature reports that time based releases are favoured in LEAN startups[21] as well as in open source projects [16]. Not only are time based schedules favoured, but very frequent releases are encouraged due to faster validation of the software. In a LEAN context this means that the MVP can iterate faster and early feedback allows the project to remain agile. For open source projects, additional benefits of minimising code divergence and a predictable schedule for external developers also exist[16].

The startups in this thesis generally agreed with the literature, the exception being Refined. They preferred a feature based schedule but still cited increased release frequency as important. In an open source context, frequent releases are somewhat trivial since new code can be pushed several times a day. When binary releases are also considered, Mapillary exhibited a higher release frequency than the rest of the startups due to continuous deployment. Bitcraze expressed an intent to do the same. Currently, binary releases were made every six months and they were not satisfied with that rate.

7.6 RQ2: Challenges

In this section the second research question are addressed while sections 7.6.1 and 7.6.2 serves as preconditions for approaching the final research question, RQ3.

(Ch XX) marks the identified challenge whereas *{Ch XX}* is a reference. This applies to the potential actions *(PA XX)* as well.

RQ2: What challenges can be identified in regards to current RE structure and practices in startups engaged in OSS development?

Elicitation The startups stated that elicitation is not an issue, it is the resources needed to implement them. In a situation where Bitcraze's BM is applied, and the startup so heavily emphasises the use of open source, this points to a broader challenge. Even if internal elicitation is sufficient to produce an impressive feature set, the community is not engaged in the development of the features or even in the discussion of what features to prioritise. The potential resources of the community are not utilised *(Ch E1)*.

Bitcraze experienced a lot of activity in their community when the product was new. While the novelty of the project spurred an initial interest, it later declined. If continual elicitation is a goal of the project, the startup needs to keep the community steadily engaged *(Ch E2)*.

Another serious issue faced by Bitcraze was that they observed new features for their project in other settings than their own. As explained by the startup; external developers actually did elicit and produce new features but they never propagated back up to the original project *(Ch E3)*.

Requirement Specification To a certain degree, the startups seemed to manage without an internal formal specification document but as the products and startups matured, the need for such a document started to present itself. Refined had already implemented one and Mapillary expressed a desire to do so. Bitcraze alluded to a need but were resistant due to their organisational culture. Regardless, a need for a more formal process seems to emerge as the startup grows. This presents two distinct challenges; when to start formalising the process *(Ch RS1)* and how to allocate resources to it *(Ch RS2)*.

One challenge faced particularly by Bitcraze was found in the bridging process. While Mapillary managed to communicate their internal issues and discussions in the open source community, Bitcraze stated that they did not do this to a desired extent *(Ch RS3)*.

Validation A potential challenge is to find trustworthy sources that are able to provide valuable feedback *(Ch V1)*.

Also, despite the majority of the startups claiming that the risk of having the project stolen is low, it still exists *(Ch V2)*, as experienced in the case of Bitcraze and how immoral manufacturers copied their platform for commercial purposes.

Moreover, the external validation is dependent on the maturity of the OS community – as in the case with Bitcraze, if the system is too complex, it could potentially impede the validation of the product *(Ch V3)*.

Release Management The challenges presented concern the frequency of releases and as it remains trivial in an open source context when only the source code is considered, we examine only the binary releases.

Both Refined and Bitcraze wanted to increase their release frequency but had not been successful so far. Since the startups differ in their release strategies this challenge can be branched into two challenges based on context.

The first challenge concerns how to increase release frequency in a feature based strategy. In Refined's case they stated that even though they had a schedule based on features, they were not happy with the frequency and as a consequence would end up with a stripped feature set *(Ch RM1)*.

The second challenge arises from the perspective of Bitcraze where a time based schedule is applied in a heavily open sourced context. They stated that even though the source code is updated often, and is always available for a client to download and build themselves, the binary releases are made every six months. The intent of the startup is to advance from a biannual schedule to daily releases *(Ch RM2)*.

7.6.1 Potential Actions to Challenges

In this section a series of potential actions that can be applied to help with the aforementioned challenges will be presented. The potential actions are based on literature as well as the interviews with the four startups. This will later serve as a component of the BOSS process presented in section 7.7.

Elicitation All three challenges presented, *{Ch E1}*, *{Ch E2}*, *{Ch E3}*, concern wasted resources from the community and Bitcraze in particular attribute them to community mismanagement, not the actual elicitation process.

The first challenge, *{Ch E1}*, can be adressed by utilising the techniques observed in many open source projects; forums and issue trackers but in addition extended to internally elicited requirements as well[24][20], in the sense that it would allow the community to discuss what features to prioritise and also outsource some of the workload to the community. This technique was successfully utilised by Mapillary, where they managed to involve the community to such extent that allowed them to fully exploit its potential, as described in section 6.2 *(PA E1)*.

The second challenge, *{Ch E2}*, implies that one cannot simply rely on the initial burst of interest that a project might experience. Employing *(PA E1)* is a good initial step, but if the startup wants to retain the community, it is our opinion that the process needs to be *continually* managed. This is supported by Bitcraze's admission of mismanagement and lack of communication and their subsequent actions to improve those aspects *(PA E2)*.

While the third challenge, *{Ch E3}*, could possibly be mitigated by *(PA E1)* and *(PA E2)*, since this is a communication problem, it might not be enough. The first two potential actions sets up a functioning environment for communication, allowing external contributors to actually contribute. However, we believe that the problem of developers not contributing back to the original project has other reasons. In Bitcraze's case, the software is spread out in 30-40 different repositories while features could potentially concern many of them. In contrast, Mapillary has not had the same issues with people not contributing, further strengthening the notion that complexity might be a concern in this challenge. If *{Ch E3}* cannot be solved by *(PA E1)* and *(PA E2)* the startup should try to re-architect the software and lower complexity *(PA E3)*.

Requirement Specification Starting by addressing *{Ch RS1}* of when to start a formal internal process we need to consider some key aspects of a startup. Because they are agile in nature, especially if the LEAN startup approach is adopted, implementing a requirements specification document too early would add friction to the iteration cycle and slow it down. On the other hand, doing it too late puts the startup at risk if key employees are absent.

One potential action is to create a requirements document after the first few rounds of feedback on the MVP. It is difficult to pinpoint a specific state of the MVP to do this as it will vary from startup to startup, but if the MVP is successful and there is no need for a pivot, the iterations should stabilise towards a final product and hence so should the specifications document *(PA RS1)*.

Another potential action is to adapt the tools currently used to include a more persistent state. As specifications are already encoded in tasks, these could easily be stored as a list of requirements, albeit primitive, with minor modifications. This would also help alleviate *{Ch RS2}* as it would not be as resource intensive considering some of the work involved is already done. This could be a very reasonable solution as evidenced by the success with informal requirements in open source projects[1] *(PA RS2)*.

As challenge *{Ch RS2}* concerns resources that startups tend to want to spend on development instead of management[21], *(PA RS2)* should be an acceptable compromise. However, considering that the real need for a formal process seems to come as a side effect of growth, the startup should recognise the importance of RE and re-allocate developer time or hire a requirements manager when the interim solution is no longer viable *(PA RS3)*.

The final challenge *{Ch RS3}* specifically concerned Bitcraze. As explained in section 6.2, they attribute this to the complexity of software, spanning over 30 repositories. This is a multifaceted challenge that could potentially be mitigated by process based, as well as technical, actions.

Bitcraze's own strategy was to create a meta repository that only contained issues with the intention of communicating features through that. Although this approach had so far been unsuccessful, the idea of creating a central space for features spanning multiple repositories is a potential action *(PA RS4)*.

Since the challenge is based on the complexity of the software, the startup could try to lower complexity. This would also help internal development. This action is the same as proposed in the elicitation section *(PA E3)*.

Finally, a potential action to *{Ch RS3}* is to adopt *selective revealing* as used by Mapillary and not release everything as open source[11]. This would make it easier for developers to concentrate their contributions while the startup can focus on supporting the community. This action has several advantages and is described further in the next section as *(PA V2)*.

Validation In order to obtain feedback the startup should not only establish good connections with their customers but also give something in return, as Bitcraze do by providing the hardware at cost, or like Refined, that attempt to alleviate the process for its users as much as possible *(PA V1)*.

Not disclosing essential information could be a difficult balancing act, especially in startups such as Bitcraze that have a defined ideology of being open. Our study suggest however, that it could be wise to follow the example of Mapillary and Refined. It would not only reduce the risk of eventually losing market shares due to theft but potentially also leads to more well-defined components, making it simpler for the developers, which in turn raise the chances for the community to develop successfully. A potential action to both *{Ch V2}* and *{Ch V3}* is therefore to separate the project into sub projects that can be validated independently *(PA V2)*.

Release Management In the setting of *{Ch RM1}* where a startup is constantly behind schedule on their feature releases there are two obvious potential actions to take. The first one is to learn from their previous mistakes and perform more realistic effort estimations. If successful, this would put releases on schedule as well as provide better cost estimates for future features*(PA RM1)*.

The second potential action is to switch to a time based strategy. This would mean that the startup is always on schedule and features are added to a release when they are done. The benefits would also include timely feedback and the customers would reliably know when to expect an update. For Refined, this is of particular interest since they do a lot of work through consultants. *(PA RM2)*.

The second challenge, *{Ch RM2}* can potentially be mitigated with a simple process action or technical actions of varying complexity. If building the software is relatively inexpensive, the startup can decide to do it manually every day *(PA RM3)*. The second potential action is to set up a build script that runs every night *(PA RM4)*. The final and most mature potential action is to employ the techniques of continuous deployment successfully used at Mapillary *(PA RM5)*.

A summary of the problems identified and solutions proposed can be found in table 2.

Table 2: Summary of problems and solutions.

Challenge	**Potential Actions**
Ch E1	PA E1
Ch E2	PA E1, PA E2
Ch E3	PA E1, PA E2, PA E3
Ch RS1	PA RS1, PA RS2
Ch RS2	PA RS2, PA RS3
Ch RS3	PA RS4, PA E3, PA V2
Ch V1	PA V1
Ch V2	PA V2
Ch V3	PA V2
Ch RM1	PA RM1, PA RM2
Ch RM2	PA RM3, PA RM4, PA RM5

7.6.2 Themes

In this section four different themes that were identified when analysing the challenges and potential actions will be presented. The themes will later be part of the BOSS Macro Analysis presented in section 7.7.3 and will be used to identify broader issues in a startup's management of an open source project.

The potential actions can all be categorised as involving changes to process, communication, technical aspects or resources. If a startup exhibits multiple challenges related to the same theme, that might indicate to the startup that special focus or further actions specific to the startup should be applied in that area. If a startup can identify the challenges presented in this thesis and the majority are, for example, communication related challenges, the startup might have broader issues relating to communication and should focus their efforts on improving communication and employ additional actions more specific to the particular startup.

The themes are further explained here.

Process If a potential action is categorised under the process theme, it means that the action involves changes to *how the startup manages certain RE tasks.* In other words, changes to current managerial practices.

Communication The communication theme indicates that a potential action will *increase communication* with the *community.* There might be a side effect of increased internal communication but the theme is primarily concerned with *external* communication towards the OSS community.

Technical While the process theme involves changes in how RE tasked are managed, the technical theme indicates that the potential action requires *changes to the technical foundation on which the process is based.* This specifically concerns what *tools*, such as issue trackers and forums, are used to carry out the RE processes as well as the software architecture and organisation of the public repositories.

Resources Potential actions with a resource theme indicates a need to *reallocate or increase capital or labour resources.*

These themes were mapped to their potential actions as displayed in table 3.

Table 3: Potential Action themes

Potential Action	**Themes**
PA E1	Process, communication
PA E2	Process, communication
PA E3	Technical
PA RS1	Process
PA RS2	Process, technical
PA RS3	Process, resources
PA RS4	Process, technical
PA V1	Process, communication, technical, resources
PA V2	Process, technical
PA RM 1	Process
PA RM 2	Process
PA RM 3	Process
PA RM 4	Technical
PA RM 5	Technical

Reasoning that because a challenge can potentially be mitigated by, for example a technical action, this might indicate that the challenge is an indicator of a broader technical issue. Applying the themes presented in table 3 to the corresponding actions in table 2 results in themes mapped to challenges presented in table 4.

Table 4: Challenges and Potential Action themes combined

Challenge	**Action themes**
Ch E1	Process, communication
Ch E2	Process, communication
Ch E3	Process, communication, technical
Ch RS1	Process, technical
Ch RS2	Process, technical, resources
Ch RS3	Process, technical
Ch V1	Process, communication, technical, resources
Ch V2	Process, technical
Ch V3	Process, technical
Ch RM1	Process
Ch RM2	Process, technical

7.7 RQ3: Engineering Solution Design of BOSS

RQ3: How can the problems identified in RQ2 be mitigated?

The third and final research question will be confronted in this section. To do this a process called Bridging Open Source and Startups (BOSS) has been designed. BOSS is a tool a startup can use to help with the process of building and leveraging an open source community. It consists of four steps as shown in figure 4.

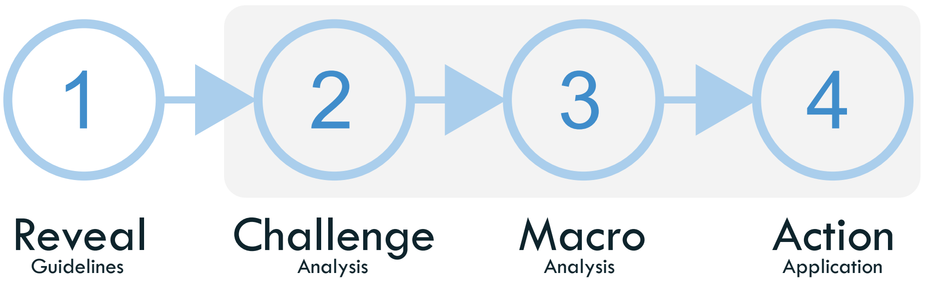


Figure 4: The BOSS method - the darker area is the iterative part of the process.

Reveal Guidelines The first step involves the release of the OSS project. BOSS contributes by facilitating some advice on attributes of the software to be released, observed side effects of releasing independent OSS projects as opposed to releasing the entire product as an OSS and a set of questions to decide what such an independent project might be. The reveal step is presented in section 7.7.1.

Challenge Analysis The second step is reached after the release of a project and is the first step of an iterative process the startup can apply during an ongoing OSS project. It involves an analysis of the current challenges faced by the startup in their OSS management process. The Challenge Analysis step is presented in section 7.7.2.

Macro Analysis While the second step is designed to examine challenges that are explicitly observable in the startup, the third step lifts the abstraction level to a macro perspective based on themes. By applying this step the startup can extract information on broader issues related to the OSS project. This step is presented in section 7.7.3.

Action Application The last step of the BOSS tool is to apply actions to mitigate the challenges identified in step 2 and 3. This thesis provides a set of potential actions to the challenges in step 2. The potential actions are based on the literature study and interviews with the startups. The details of this step are given in section 7.7.4.

The BOSS tool is an agile process, allowing the startup to quickly get insight and help with the initiation of the OSS project with step one. It is also an iterative process and once the project is up and running the iteration can start with the second step. Throughout the lifetime of the OSS project, the startup can return and reiterate through steps two, three and four intermittently as needed. The iterations are needed as the startup and the environment it operates in changes quickly and as such the outcomes of the steps may change with each iteration.

The BOSS tool was designed by exploring a variety of hypotheses on how to best help a startup with an OSS project. These hypotheses largely resulted in minor tools, drawing inspiration from classical management tools such as SWOT analysis and Six Forces. The goal was to aid in decision making based on a variety of parameters such as community maturity, desired involvement, complexity, etc. The tools were ultimately too scattered, abstract and with questionable rationale. In the end, a conclusion was made that the tool needed to include concrete challenges and potential actions as a counter measure to experienced challenges. This would provide a far more pragmatic approach and be helpful to a user regardless of previous managerial experience. Realising however that the current state of a startup and their priorities may change rapidly, simply providing a list of challenges and potential actions is not enough. The tool needed a focus mechanism to help a startup evaluate what challenges to prioritise at a given moment. Additionally, the list of challenges in this thesis is of course non exhaustive as one startup's situation may provide more unique challenges. This lead the design process to two realisations; the tool needed to be iterative because of the continuously changing environment and it also needed an additional mechanism to provide help tailored to the operating startup's unique situation. The iterative approach is also compatible with the LEAN startup methodology and other agile methodologies used by startups. Ultimately this meant a three step iterative design; a Challenge Analysis, a Macro Analysis and an Action Application step. The Macro Analysis is based on four different themes identified as general enough to provide a compass for a startup to quickly search for other challenges not found in this thesis. The combination of the iterative approach and Macro Analysis would provide a simple way for a startup to always focus on the most pressing challenges while also drawing attention to areas of immediate importance. Finally, to be truly useful for new startups and new projects the tool needed an inception stage. The knowledge extracted from this thesis was compiled into the Reveal Guidelines step. This step needed to be very lightweight to enable startups to release quickly. It is useful for all types of startups since it incorporates advice on software qualities, a view on releasing a product as opposed to single components as well as a simple way of identifying suitable components.

What ultimately resulted from the design process is the BOOS tool presented in this thesis. It is an agile and iterative tool a startup can use for guidance throughout an OSS venture. The tool is also naturally extensible and further research can be done to find new challenges and potential actions.

7.7.1 BOSS Revealing Guidelines

The first step of the BOSS process is designed to quickly help the startup with a major decision; what to release as an OSS project. In section 7.3 results of the literature study and interviews are presented and in this step they are compiled as compact general guidelines.

The first decision to make is whether a particular project or product is suitable for an OSS release. The following are some beneficial preconditions that will help if contributions are desired.

Demand There should be a public demand for the project and it should solve a problem.

State While good software quality is beneficial, the important aspect is that developers can test it and it convincingly demonstrates *potential.*

Novelty The project can be new and innovative, but it can also present new features or improvements over competing projects or be an OSS alternative to proprietary software.

The second decision to make is whether to release an entire product as OSS, or to release smaller components as independent projects. The first situation seems to be applicable if there are ideological reasons for making an OSS product or if the product is to serve as an OSS alternative to a commercial competitor. The latter situation should be applicable in any software startup.

The following side effects were observed when multiple independent projects were released, as opposed to one large project.

IP More granular control of intellectual property.

Prioritisation The startup can prioritise between different projects and apply different levels of involvement accordingly. This also means that some components can be released for low effort reasons such as branding while other projects can be released with contribution goals and have different RE frameworks.

Risk of theft Releasing the entire product as OSS puts the startup at an obvious risk for competing forks. If separate components are released the product cannot be copied without reverse engineering the proprietary parts.

Changing licenses If a license needs to be changed to a more restrictive license after external contributions have been made, permission is needed from all contributors. Smaller independent projects may lower exposure and spread the risk.

These side effects may seem biased towards the component model but are the results of the analysis in this thesis. Further research can explore what benefits a product model may have over a component model.

Finally, if the decision to release separate components as OSS is made, the following questions will help identifying such a component.

1. Does this component have a clear purpose?

2. Is it non-critical for the product?

3. Does it have to be maintained?

4. Is it reusable?

This first step of the BOSS process should be done once for every new OSS venture. After the release of the project the iterative process of steps 2,3 and 4 can be initiated.

7.7.2 BOSS Challenge Analysis

The BOSS Challenge Analysis is the second step of the BOSS process and the first step of the iterative part. This step involves an analysis of the management of the OSS project from the RE perspectives of elicitation, requirements specification, validation and release management. The identification process can be done with the checklist presented in table 5. For the benefit of startups not versed in RE terminology the RE areas have been replaced with simplified categories.

Table 5: Challenge identification

Id	**Difficulties in obtaining ideas and solutions.**	**Check**
Ch E1	Even if features are elicited internally, the community is not involved in discussions and prioritisation of them.	
Ch E2	While novelty of a project might spur initial interest, continuing to keep the community steadily engaged in the elicitation process is difficult.	
Ch E3	New features can appear in forks or closed projects and not propagate back to the original project.	
Id	**Difficulties in keeping track of ideas and solutions.**	
Ch RS1	As a project grows, the need for formal specification grows with it and deciding when to start formalising a process is difficult.	
Ch RS2	As a project grows, the need for formal specification grows with it and deciding how to allocate resources to the process is difficult.	
Ch RS3	Communicating internal issues and discussions to the OSS community is not done to a desired extent.	
Id	**Difficulties in obtaining valuable feedback.**	
Ch V1	Finding sources that can provide valuable feedback.	
Ch V2	Trying to validate the project also puts the startup at risk for theft.	
Ch V3	If the project is very complex it might require a very mature OSS community and could impede validation.	
Id	**Difficulties related to the release process.**	
Ch RM1	Increasing release frequency in a feature based release schedule without constantly ending up with a stripped feature set.	
Ch RM2	Increasing release frequency of binary releases of an OSS project.	

The challenges presented here in table 5 are summarised for convenience and presentation reasons but the full context and reasons for derivation can be seen in section 7.6. Once relevant challenges have been identified, step 2 can be initiated. The data gathered in this step will be used in the checklist that is part of the next step.

7.7.3 BOSS Macro Analysis

This section presents the third step of the BOSS process. The application of this step consists of applying the tool presented in table 7. While the previous step will help with identifying specific challenges to a startup's open source management practices, this step will identify broader issues relating to process, communication, technology and resources.

If we extract the four previously mentioned themes detailed in section 7.6.2 and put them in individual columns we can generate a mapping matrix. Table 6 is another way of representing the same information presented in table 4. If we consider *Ch E1*, it has been mapped to *process* and *communication* in table 4 and hence the same themes are marked with a ✓in the row of *Ch E1* in table 6.

Table 6: Mapping matrix

Challenge	*Process*	*Communication*	*Technical*	*Resources*
Ch E1	✓	✓	-	-
Ch E2	✓	✓	-	-
Ch E3	✓	✓	✓	-
Ch RS1	✓	-	✓	-
Ch RS2	✓	-	✓	-
Ch RS3	✓	-	✓	-
Ch V1	✓	✓	✓	✓
Ch V2	✓	-	✓	-
Ch V3	✓	-	✓	-
Ch RM1	✓	-	-	-
Ch RM2	✓	-	✓	-

If we further extend the mapping matrix in table 6 with a *Checklist* column and a summarising row, named *Checked* we get the tool presented in table 7.

Application The final tool is showcased with a filled in example of how a finished checklist would look after it has been used by a startup. The example and usage of the tool is further discussed in the paragraph following the table.

Table 7: Checklist example

Challenge	*Process*	*Communication*	*Technical*	*Resources*	**Checklist**
Ch E1	✓	✓	-	-	✓
Ch E2	✓	✓	-	-	✓
Ch E3	✓	✓	✓	-	-
Ch RS1	✓	-	✓	-	-
Ch RS2	✓	-	✓	✓	-
Ch RS3	✓	-	✓	-	-
Ch V1	✓	✓	✓	✓	✓
Ch V2	✓	-	✓	-	-
Ch V3	✓	-	✓	-	-
Ch RM1	✓	-	-	-	-
Ch RM2	✓	-	✓	-	-
Checked	**3**	**3**	**1**	**1**	

In this example, the ✓ indicates that *Ch E1*, *Ch E2*, *Ch V1* have been identified. The checked sum at the bottom is the number of challenges identified that are associated with that particular theme column. The example input indicates that the fictive startup might have more general issues related to *process* and *communication* because all three challenges identified were associated with those themes. Conversely, there is only one identified challenge associated with the *technical* and *resources* themes, indicating that they are of comparatively lesser concern to the startup. In addition to evaluating and possibly employing the potential actions suggested in this thesis, the startup should evaluate their *process* and *communication* more closely and implement improvements more specific to their particular needs. The identification of critical themes will be valuable when prioritising between different actions in the next step.

7.7.4 BOSS Action Application

The final step of the BOSS process is to apply actions to mitigate the challenges identified in step 2 and 3.

For challenges identified in step 2, BOSS Challenge Identification, this thesis provides the potential actions described in section 7.6.1. Table 8 provides the mappings of challenges to relating potential actions. In table 9 brief summaries of the actions are given for presentation reasons but the full context should be reviewed in the previously mentioned section 7.6.1. A summary of the challenges was given in table 5 and the full context of the challenges is found in section 7.6.

Table 8: Mapping of challenges and actions.

Challenge	**Potential Actions**
Ch E1	PA E1
Ch E2	PA E1, PA E2
Ch E3	PA E1, PA E2, PA E3
Ch RS1	PA RS1, PA RS2
Ch RS2	PA RS2, PA RS3
Ch RS3	PA RS4, PA E3, PA V2
Ch V1	PA V1
Ch V2	PA V2
Ch V3	PA V2
Ch RM1	PA RM1, PA RM2
Ch RM2	PA RM3, PA RM4, PA RM5

Table 9: Summary of potential actions

PA Id	**Brief Summary**
PA E1	Utilise forums and issue trackers for internally elicited requirements.
PA E2	Be active and continually communicate with and manage the OSS community.
PA E3	Re-architect the project to lower complexity.
PA RS1	Establish a requirements specification document after the first few rounds of a successful MVP
PA RS2	Perform minor modifications to tasks used internally and use them as primitive specifications.
PA RS3	As the need for specifying requirements seem to be a function of growth, plan to hire a person that can be tasked with requirements management. Alternatively, re-allocate developer time.
PA RS4	Create a meta repository specifically for broader features and issues.
PA V1	Establish good connections with customers. Give something in return for validation, e.g hardware at cost or spend time simplifying the process as much as possible.
PA V2	Separate components into independent sub projects that can be validated individually.
PA RM1	Learn from previous mistakes to perform more realistic effort estimations.
PA RM2	Switch to a time based release strategy.
PA RM3	If the build process is inexpensive, do it manually every day.
PA RM4	Set up an automated build script that runs every night.
PA RM5	Employ continuous deployment.

To perform this final step of the BOSS process the startup needs to do the following:

1. Use the data acquired in step 2, BOSS Challenge Identification, to extract potential actions for the challenges.

2. Evaluate the identified potential actions and their applicability for the startup.
3. Apply the potential actions. If the identified potential actions are too many, focus on the most critical theme identified in step 3, BOSS Macro Analysis.
4. Apply additional actions more specific to the individual startup's situation with a focus on the most critical theme identified in step 3, BOSS Macro Analysis.

After applying this step, the startup will have finished their first iteration of the BOSS process. Because the process is iterative, not all identified actions need to be applied immediately. It is recommended to implement a subset of the actions and evaluate the result with the next iteration. Because startups evolve rapidly and their priorities and relation to their OSS communities might change it is important to return to the BOSS process intermittently as challenges not identified during one iteration might present themselves later. By focusing on the themes identified in step 3, BOSS Macro Analysis, the startup can prioritise actions in areas that are of particular importance during the current iteration.

8 Validation

In order to truly evaluate the BOSS tool an unorthodox approach was undertaken: A website consisting of six slides (screenshots can be found in the appendix) was built by the authors to be able to fully demonstrate the outcome of this thesis. The technology used to build the frontend was a mixture of HTML and CSS, jQuery/Javascript, Gulp, node.js and Adobe Illustrator. The backend whose purpose was to assess and store the data collected from the participants was constructed with jQuery/Javascript, Google Analytics and Google Firebase. The presentation was essentially divided into two steps, namely, a presentation of the tool and a questionnaire. Below are detailed descriptions of each slide:

First slide The first slide presented the entire purpose of the validation part and the BOSS tool as a whole. It also provided an overview of the two parts – presentations of the tool and questionnaire – as mentioned above. Lastly, a login animation above the first header indicated whether the user obtained a valid token. The token, which was incorporated in the link sent to the participants, is a part of the Google Firebase installation and is required in order to be able to assess and store.

Second slide This part outlined the BOSS tool with its four steps. It also explained what it is and how it works.

Third slide This is a questionnaire specifically designed to validate the Reveal Guidelines. The participants were asked to toggle the switches they believed to be true. The statements were the following:

S1: Many small projects means more granular control of IP.

S2: Many small projects means more granular control of licensing.

S3: Many small projects means easier to prioritise between the different projects.

S4: It is harder for someone to copy a product if the product also includes proprietary software.

S5: Changing licensing of an already released open source project with external contributors can be difficult.

The response to these statements can be seen in table 10. In all following tables showcasing the responses from the validation, the respondents are marked R1, R2, R3 and R4 respectively.

Table 10: Reveal Guidelines Response

Statement	**R1**	**R2**	**R3**	**R4**
S1	Yes	Yes	No	Yes
S2	Yes	Yes	No	Yes
S3	No	No	No	Yes
S4	Yes	Yes	No	Yes
S5	Yes	Yes	Yes	Yes

Overall, all statements had a majority agreement from respondents expect for statement S3. Unfortunately no explanation as to why was given but we believe this particular statement was not properly communicated. Had the full context been given, as presented in this thesis, we believe the answers would have been different.

Fourth slide This was an interactive section that not only allowed the participants to explore BOSS Challenge Analysis and its components, challenges and potential actions but to comment as well. This was done by simply hovering the mouse over each challenge or click, respectively.

The responses from this section can be seen in table 11.

Table 11: Challenge and potential action response

Challenge	**R3**
Even if features are elicited internally, the community is not involved in discussions and prioritisation of them.	True for us. We might not be that good at making discussions public (it often happens that we discuss things in the office) but we have also noticed that it might be hard to get the community to participate in the discussions.
Increasing release frequency of binary releases of an OSS project.	My suggestion is to start with TTD, CD and other practices that forces every one to work in a orderly way as soon as possible.

Unfortunately only one participant, R3, gave any comments on the challenges and actions. It is unclear if the reason is due to a general agreement on the challenges and actions or due to time constraints. However, the comments given are valuable. We interpret the first comment as meaning that even if internal features are exposed in external issue trackers, the community might still not engage. This is of course true and hence the suggested actions are termed 'Potential actions'. It is more clearly communicated in the context of this thesis and should be considered before the next time the BOSS tool is presented in a summarised form. The last suggestion is interesting and further research can explore at what time such implementations are worth considering.

Fifth slide This slide presented BOSS Macro Analysis and Boss Action Application. It also provided a recap of the entire process, outlining the iterative process both visually and textually.

Sixth slide The participants were able to provide feedback if they did not agree with any of the statements in BOSS Reveal Guidelines, if they had any comments on Boss Macro Analysis or Boss Action Application and general comments as well. The comments are presented in table 12.

Table 12: Comments

Comment type	**R3**
Reveal Guidelines	Most of the points are related to IP and licensing. I think technical reasons could be another good reason for many small projects, see micro services for instance.
Macro / Actions	Macro analysis: You will probably always have all the problems at the same time to some degree. Maybe the instruction for step 2 should be P̈ick the 3 biggest challenges you have right now.̈ I like the iterative approach as it goes hand in hand with many other methods/ideas such as scrum, lean startup, evolutionary development and so on. It might be beneficial to add some suggestions on the length of the iteration cycle?
General	Well done!

Again, only respondent R3 had any comments to give. As with the challenge responses, this could be due to a general agreement or time constraints. We interpret the first comment as meaning that smaller projects, as opposed to a large product, can have technical benefits. This is a good idea and could be explored in further research.

The second comment is great. It essentially means that the respondent agrees that the challenges are plausible. Prioritisation between challenges is already included in the Macro Analysis step. This is most likely better communicated in this thesis and should be considered the next time the BOSS tool is presented. It is positive to see that the iterative approach is appreciated and for the precise reasons it was implemented; challenges and their importance change over time and iteration is compatible with other methodologies used by startups.

The last question asked was wether the respondents thought the BOSS tool could be helpful to other startups seeking to leverage OSS communities. The response can be found in table 13.

Table 13: Helpfulness

Question	**R1**	**R2**	**R3**	**R4**
Do you think that the BOSS tool could be helpful to other startups seeking to leverage OSS communities?	Yes	No	Yes	Yes

Three of the four respondents believed that the BOSS tool could be helpful to a startup. It is possible that R2 did not have time to fully understand the tool or genuinely does not see it as useful, however, no comments were given so the reason remains unknown.

Overall the response was very positive and is a good indicator that the BOSS tool has potential promise in practice.

9 Threats to Validity

This section presents validity threats concerning this thesis. Following threats have been identified [23]: Internal validity, external validity, construct validity and reliability.

9.1 Internal Validity

This deals with the mitigation of potential bias. Triangulation was used in the form of multiple literature sources. However, the insufficiency of academical papers in the subject that this thesis covers impeded the process to completely eradicate bias. In addition, the authors' different viewpoints were used in an attempt to be as objective as possible.

9.2 External Validity

This concerns with to what degree the results can be generalised to other contexts – that in this case remains unclear to what extent the findings in this thesis can be. On one hand some of the interviewed startup had tremendous experience and track record, combined with the fact that their way of conducting RE many times resonated with the literature found. However, there seem to exist several different strategies on how to build communities which could potentially hinder the possibility of a generalisation. Further research needs to be done in order to attain a broader perspective.

9.3 Construct Validity

This regards to the selection of the correct measures for the research. Only startups involved in OSS communities were selected. The interviews were semi-structured in order to be able to adjust the questions, to limit any potential bias. The questions were often repeated in other shapes, such as with different examples and formulations. The interview questions were carefully crafted by analysing and understanding the process cycle of a startup and how open source in works in general.

9.4 Reliability

This refers to what extent this study can be replicated. In order to enhance the traceability and reliability, all interviews were recorded and transcribed. Also, a form was sent out to the startups with the purpose to further validate the results.

10 Discussion

It was expected to see very informal RE processes in both the startups and OSS projects. This was true with all the startups in this thesis and has been observed in previous research indicating that this is more likely the norm than exception. It was however interesting to see that the processes are informal in different ways and ultimately not immediately compatible. They can however be separated in an internal and an external process that can vary independently. The sample size of the interviewed startups is small but covers many different situations. The startups vary with success rate, maturity as a company, complexity of both the product and OSS projects as well as business models, OSS strategies and OSS involvement. Although these parameters are represented by the startups, the sample size is too small to attribute any general conclusions with regard to any specific parameter.

We expected the OSS projects to be more successful and this probably has more to do with a selection bias of previous research where studies have mostly been done on already successful OSS projects. In this thesis we observed early stage OSS projects that were released for different reasons, possibly requiring other metrics of success than number of contributions and adoption by developers.

Startups and OSS projects share many process similarities and ideas such as minimising management overhead and adopting agile philosophies, especially when compared to how a larger corporation operates. Combined with the fact that startups are typically very resource constrained, reliant on innovation and that OSS might offer a suitable solution to both those problems one might think that merging startups with OSS is a perfect match. In this thesis we have found that there are a variety of different ways to implement the combination and it is not without challenges specific to this situation. It is simply not enough that both startups and OSS have similar philosophies and typically rely on very informal RE practices. In fact, sometimes they are very contradictory in the implementation of the processes.

While a startup might enjoy freedom from management overhead through less documentation and a very informal RE process, a major reason why this works is most likely that the startup operates as a very tight knit team with very efficient communication pathways and a deep understanding of the project that is being developed. An OSS project on the other hand typically does not have the luxury of very committed team members that will stay in the project for a long time. Instead, those projects rely on ease of contribution and more often than not have few contributions from many developers. This means that apart from the issue of motivating an external developer, it needs to be easy to quickly understand a specific part of the project and contribute to it. This is largely done through documentation of the project and is typically much more thorough than what is needed in a constantly evolving project where team members are well versed with the code base. The contradictions do not mean that the combination of startups and OSS is completely incompatible and should not be pursued, there are ways of successfully leveraging positive aspects of both and this is especially evidenced by Mapillary and Neo in this thesis. We believe a reason as to why this is might be attributed to the selective revealing aspect that was employed at Mapillary, Neo and Refined. This is contrary to releasing all source code as done by Bitcraze.

By releasing only well selected parts of the code base the startup can employ separate processes for proprietary development and OSS development. The internal process can develop and mature as needed by the startup while the OSS projects can be managed according to the best practices already established in the OSS community. Releasing certain parts as OSS will require more documentation and communication with external developers, but the extent to which the startup chooses to do this can be adjusted according to their current needs. It also has the potential to mitigate risk of theft by not exposing the entire code base of the product. Releasing separate components as OSS most likely also requires some preparation of the code if the OSS project is to succeed in attracting developers which would likely increase code quality. The extra requirements in more external communication, management, documentation and code quality will of course be a resource cost for the startup, but those are costs of any successful OSS project so employing selective revealing can focus those resources towards areas of the startup's choice. If the entire code base is released as OSS those costs will affect a larger code base. Varying the level of engagement in separate projects could potentially be very valuable. Mapillary showed that if contributions are the main purpose of an OSS release, greater effort can be put into documenting that project while Refined released a project mainly for branding purposes, requiring little to no effort in the release process and documentation of the project. Different projects can have different purposes for an OSS release where low effort reasons include branding, IP protection, bug reporting and feature requests. Higher effort purposes would include bug fixes and feature implementations.

The positive aspects of selective revealing do not necessarily mean that releasing the entire code base is a bad idea. From what was observed in this thesis however, the particular combination of extremely informal RE practices combined with a very complex and large code base did not ultimately yield a satisfactory result. Bitcraze's aspirations of having as little documentation as possible internally did not translate well to the open source nature of the startup. Although they did make a big point of being as open as possible that does not simply mean releasing all the code, they need to actively communicate what they are working on and what their plans for the project are as well as be responsive to input from the community. In Mapillary's case they would treat all internal development within the OSS projects as if they were external, something Bitcraze aspired to but did not actually do. As OSS is such an important aspect for Bitcraze's product, they should probably treat the external community as part of the startup. They should be equal to the members of the internal development team, meaning that Bitcraze should treat their development process as if they had a very large number of remote developers.

Bitcraze's and Neo's models are perhaps the most interesting for further research. It is a model where a startup releases the entire, or a vast majority, of the code base as OSS. It has the potential to alleviate a lot of the resource constraints naturally put on startups as well as offer a good source for innovation. There has been a lot of research on OSS projects of varying sizes but those are largely non-profit projects or sponsored by large companies. It would be interesting to if there is a certain type of product that is suitable for this 'OSS startup' model. Infrastructure and tooling for developers seems promising as Neo4j is a database and the Crazyflie is a development platform. As Bitcraze initially had a lot of interest and a fair amount of contributions it would be interesting to do a follow up study if they improve their processes. Further research could also include how these 'OSS startups' should monetise their products. Neo was obviously very successful in employing the dual licensing model but it might require special consideration as to what license should be used. If a dual licensing model is applied, it would be interesting to see at what point it should be introduced. Applying it from the start could potentially hinder external contributions and spawn a competing free product while doing it too late requires careful licensing planning or explicit permission from all contributors at that point.

The method of selective revealing employed at the other startups is potentially applicable to any software startup. Identify a component that is, as excellently put by Neubauer, *'a clearly separate component that has its own purpose and is not critical to my project, it's more of a by-product that I also have to maintain and is reusable'* , release it and tailor the process according to the need for contributions. At this point the research on OSS in general should be applicable and research on startups could be applied internally. It is a much clearer separation of the startup and an OSS project. It would however be interesting to find out if there is a more well defined selection process for such components that also incorporates the current state of the startup and maturity of the component.

From a pure RE point of view it would be interesting to see if there are other processes and tools that can be developed specifically for the 'OSS startup' model. This would mean a unified process and toolset for internal and external development that satisfies both the internal desire to keep overhead in the form of documentation and formal RE processes from slowing down development while also enabling remote external developers to easily understand the code base, follow internal progress and contribute to the product development process by taking part in RE tasks and contribute to the code base.

11 Conclusions

In this thesis we have answered three research questions:

RQ1: How is RE currently structured and practiced in startups engaged in OSS development?

RQ2: What challenges can be identified in regards to current RE structure and practices in startups engaged in OSS development?

RQ3: How can the problems identified in RQ2 be mitigated?

We have presented a literature study on the state of requirements engineering as practiced in startups and open source projects respectively. The results in the literature study have been compared with data from interviews with four local startups on how they do open source requirements engineering. The combined results of the literature study and analysis of the interviews were used to gain an understanding of the current state of RE within the intersection of OSS and startups. RQ1 is answered from the perspectives of elicitation, specification, validation and release management in section 7.5. We conclude that RE in OSS and startups are both informal when compared to classical RE practices but are not immediately compatible. We believe this is largely due to differing communication needs. More specific results concerning the four RE areas includes among others; a view on internal elicitation as a scope setting mechanism for the startup, difficulties concerning the communication of internal specifications, the OSS community as a pool of validation sources and a firm focus on release frequency.

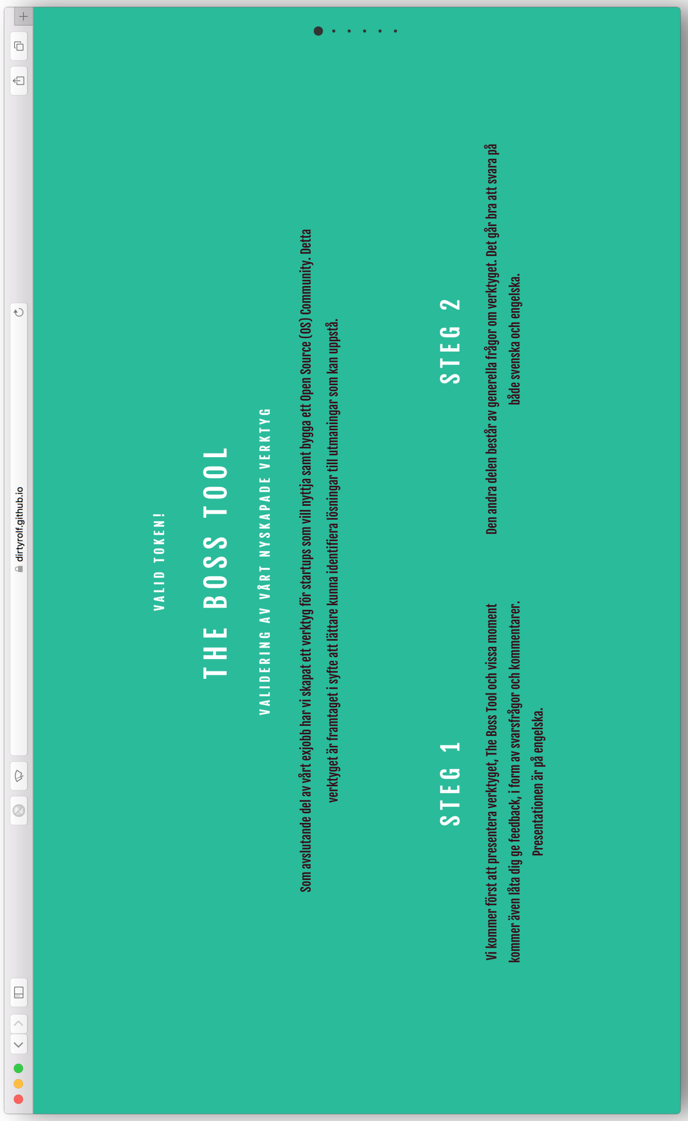
The second research questions concerns challenges experienced by the startups in the context of requirements engineering. Eleven challenges have been identified from our interviews with the four participating startups and are categorised in the four main RE areas mentioned earlier. The full set of challenges can be found in section 7.6 and brief summaries are given in table 5 in section 7.7.2, page 51.

The challenges are addressed in the third research question by the BOSS tool which we have developed in this thesis and is found in section 7.7. BOSS is an acronym for Bridging Open Source & Startups and is a tool designed to quickly and iteratively help a startup with an OSS venture. It consists of four distinct steps; Reveal Guidelines, Challenge Analysis, Macro Analysis and Action Application. The first step quickly helps a startup with critical decisions related to the release of an OSS project while the latter three iteratively helps the startup with identifying challenges, focusing on what is currently important and taking action to mitigate the challenges. The last step includes 14 potential actions designed in this thesis and are based on interviews and literature. The BOSS tool is expandable and further research can be done to identify a larger set of challenges and potential actions.

In summation; we investigate the current state of RE within the boundaries of a startup engaged in an OSS release (section 7.5), identify eleven challenges in this context (section 7.6) and present a tool to aid a startup in the venture (section 7.7).

Appendices

A Screenshots of validation webpage



VALID TOKEN!

THE BOSS TOOL

VALIDERING AV VÅRT NYSKAPADE VERKTYG

Som avslutande del av vårt exjobb har vi skapat ett verktyg för startups som vill nyttja samt bygga ett Open Source (OS) Community. Detta verktyget är framtaget i syfte att lättare kunna identifiera lösningar till utmaningar som kan uppstå.

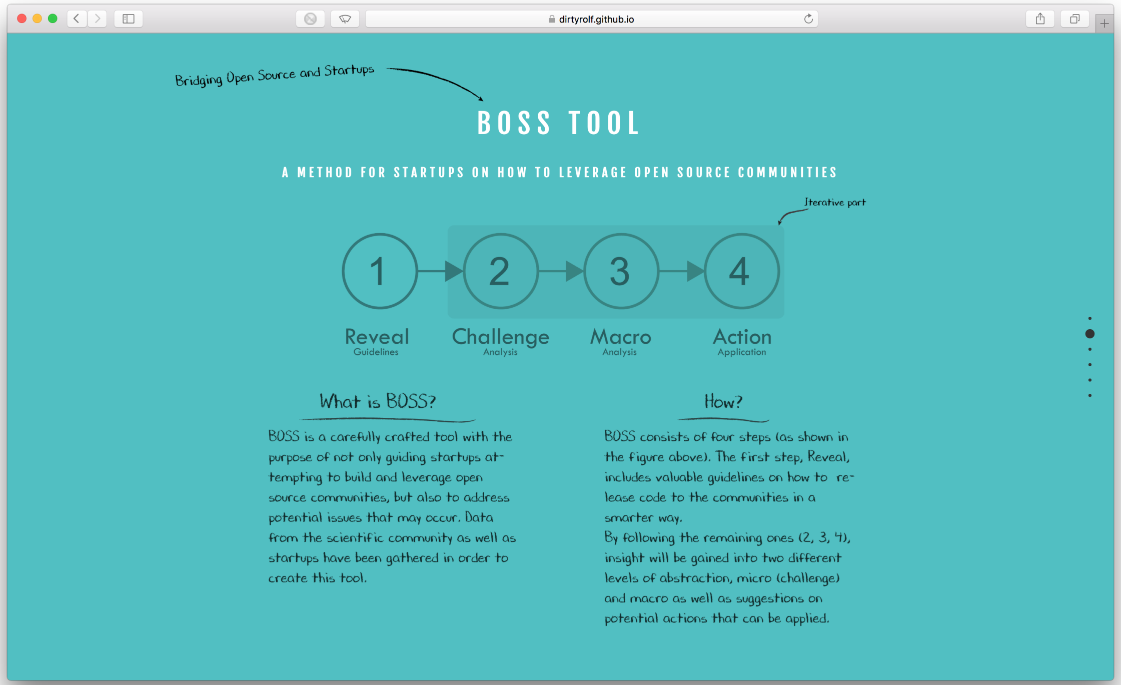
STEG 1

Vi kommer först att presentera verktyget, The Boss Tool och vissa moment kommer även låta dig ge feedback, i form av svarsfrågor och kommentarer. Presentationen är på engelska.

STEG 2

Den andra delen består av generella frågor om verktyget. Det går bra att svara på både svenska och engelska.

Figure 5: Presentation of the validation process in general



dirtyrolf.github.io

Bridging Open Source and Startups

BOSS TOOL

A METHOD FOR STARTUPS ON HOW TO LEVERAGE OPEN SOURCE COMMUNITIES

Iterative part

1 Reveal Guidelines → 2 Challenge Analysis → 3 Macro Analysis → 4 Action Application

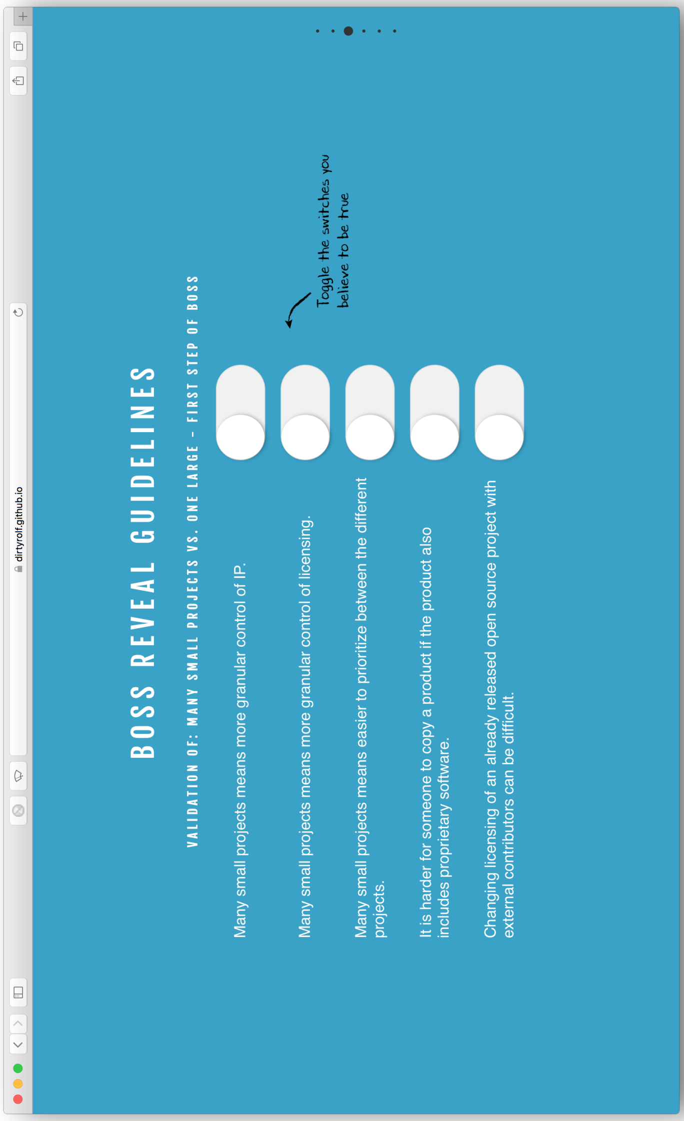
What is BOSS?

BOSS is a carefully crafted tool with the purpose of not only guiding startups attempting to build and leverage open source communities, but also to address potential issues that may occur. Data from the scientific community as well as startups have been gathered in order to create this tool.

How?

BOSS consists of four steps (as shown in the figure above). The first step, Reveal, includes valuable guidelines on how to release code to the communities in a smarter way.
By following the remaining ones (2, 3, 4), insight will be gained into two different levels of abstraction, micro (challenge) and macro as well as suggestions on potential actions that can be applied.

Figure 6: The BOSS tool - Overview



BOSS REVEAL GUIDELINES

VALIDATION OF: MANY SMALL PROJECTS VS. ONE LARGE – FIRST STEP OF BOSS

Toggle the switches you believe to be true

Many small projects means more granular control of IP.

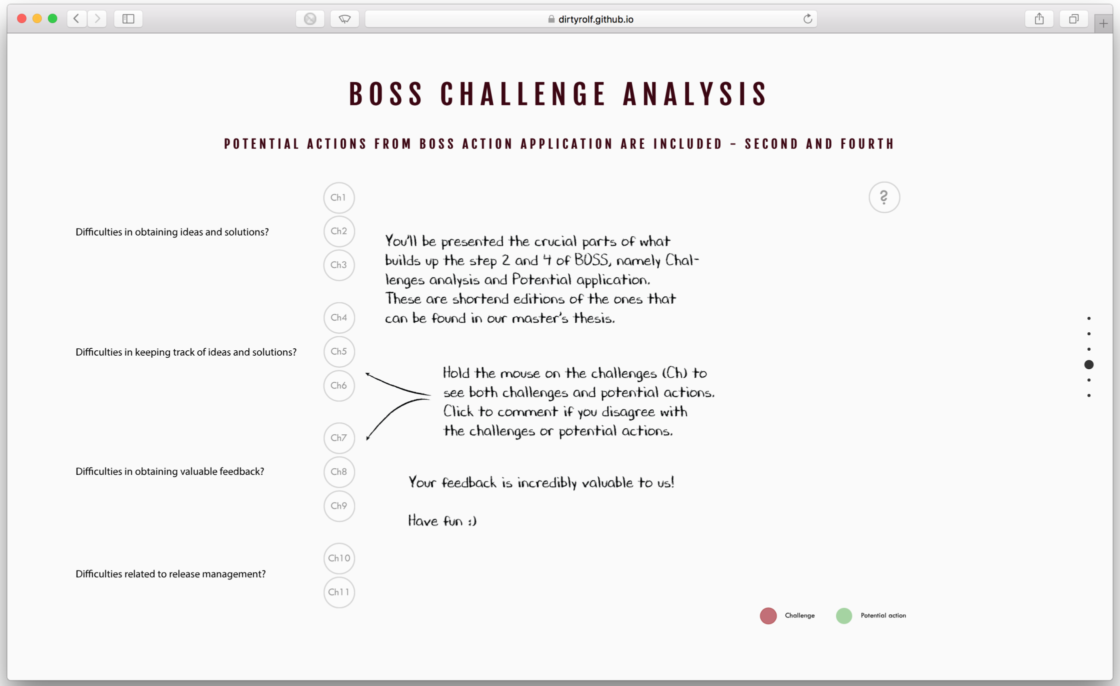
Many small projects means more granular control of licensing.

Many small projects means easier to prioritize between the different projects.

It is harder for someone to copy a product if the product also includes proprietary software.

Changing licensing of an already released open source project with external contributors can be difficult.

Figure 7: BOSS Reveal Guidelines



BOSS CHALLENGE ANALYSIS

POTENTIAL ACTIONS FROM BOSS ACTION APPLICATION ARE INCLUDED – SECOND AND FOURTH

Difficulties in obtaining ideas and solutions?

Difficulties in keeping track of ideas and solutions?

Difficulties in obtaining valuable feedback?

Difficulties related to release management?

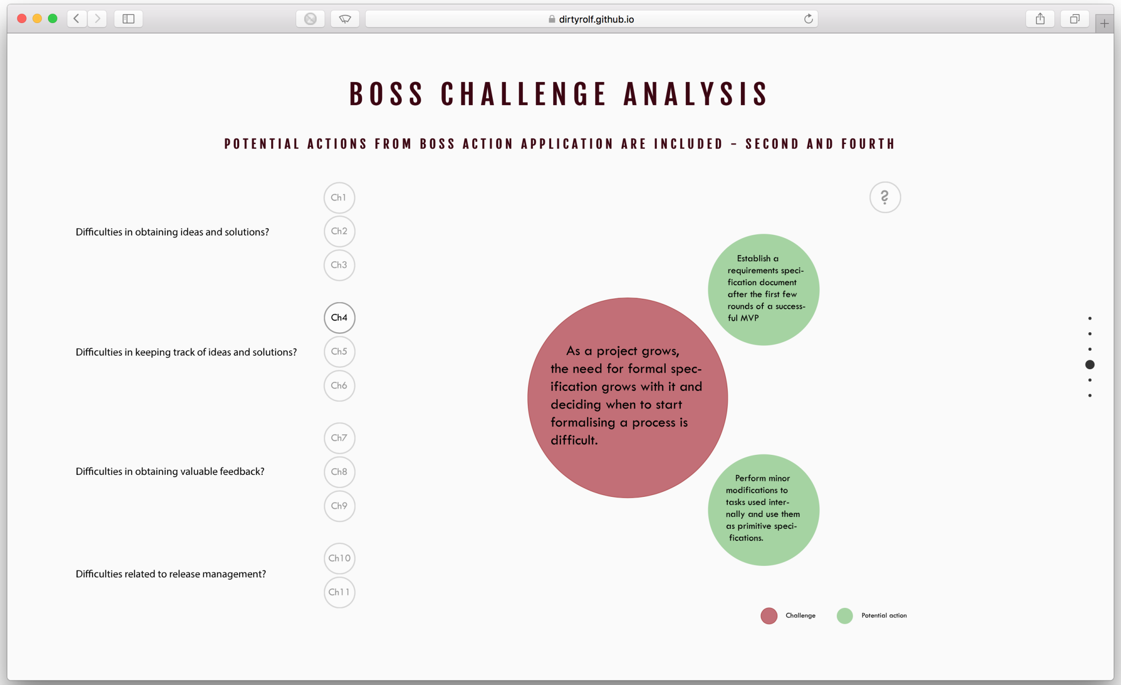
You'll be presented the crucial parts of what builds up the step 2 and 4 of BOSS, namely Challenges analysis and Potential application. These are shortend editions of the ones that can be found in our master's thesis.

Hold the mouse on the challenges (Ch) to see both challenges and potential actions. Click to comment if you disagree with the challenges or potential actions.

Your feedback is incredibly valuable to us!

Have fun :)

Figure 8: BOSS Challenge Analysis without hover/click



dirtyrolf.github.io

BOSS CHALLENGE ANALYSIS

POTENTIAL ACTIONS FROM BOSS ACTION APPLICATION ARE INCLUDED – SECOND AND FOURTH

Difficulties in obtaining ideas and solutions?

Ch1 Ch2 Ch3

Difficulties in keeping track of ideas and solutions?

Ch4 Ch5 Ch6

Difficulties in obtaining valuable feedback?

Ch7 Ch8 Ch9

Difficulties related to release management?

Ch10 Ch11

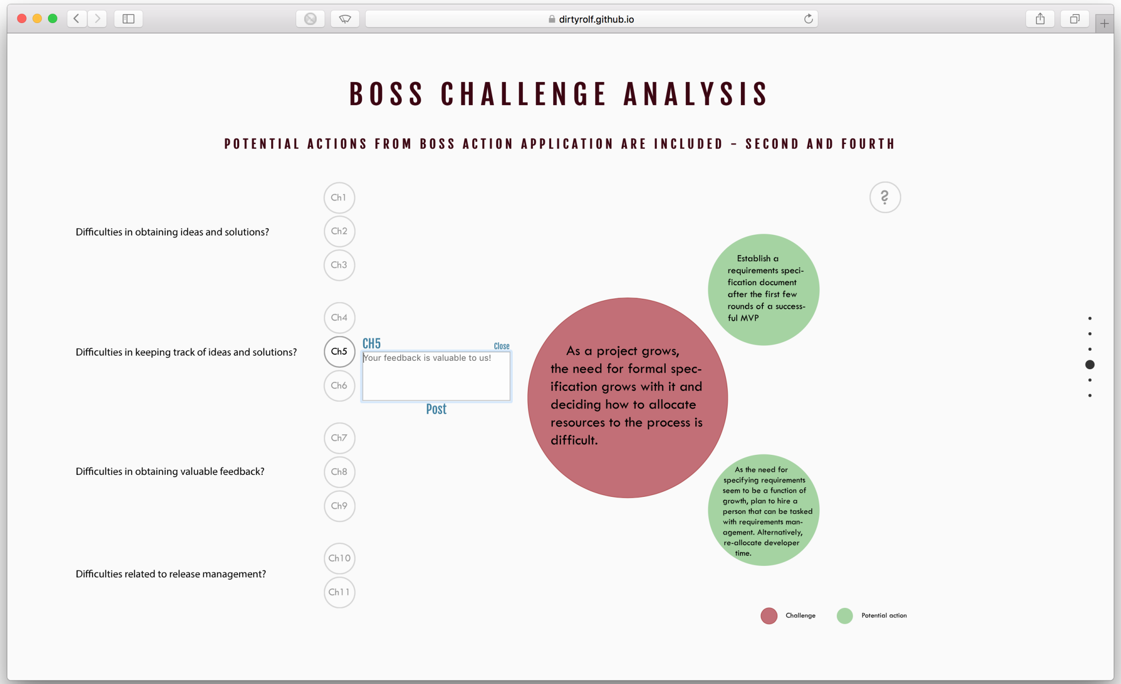
As a project grows, the need for formal specification grows with it and deciding when to start formalising a process is difficult.

Establish a requirements specification document after the first few rounds of a successful MVP

Perform minor modifications to tasks used internally and use them as primitive specifications.

🟥 Challenge 🟩 Potential action

Figure 9: BOSS Challenge Analysis without hover



BOSS CHALLENGE ANALYSIS

POTENTIAL ACTIONS FROM BOSS ACTION APPLICATION ARE INCLUDED – SECOND AND FOURTH

Difficulties in obtaining ideas and solutions?

Difficulties in keeping track of ideas and solutions?

Difficulties in obtaining valuable feedback?

Difficulties related to release management?

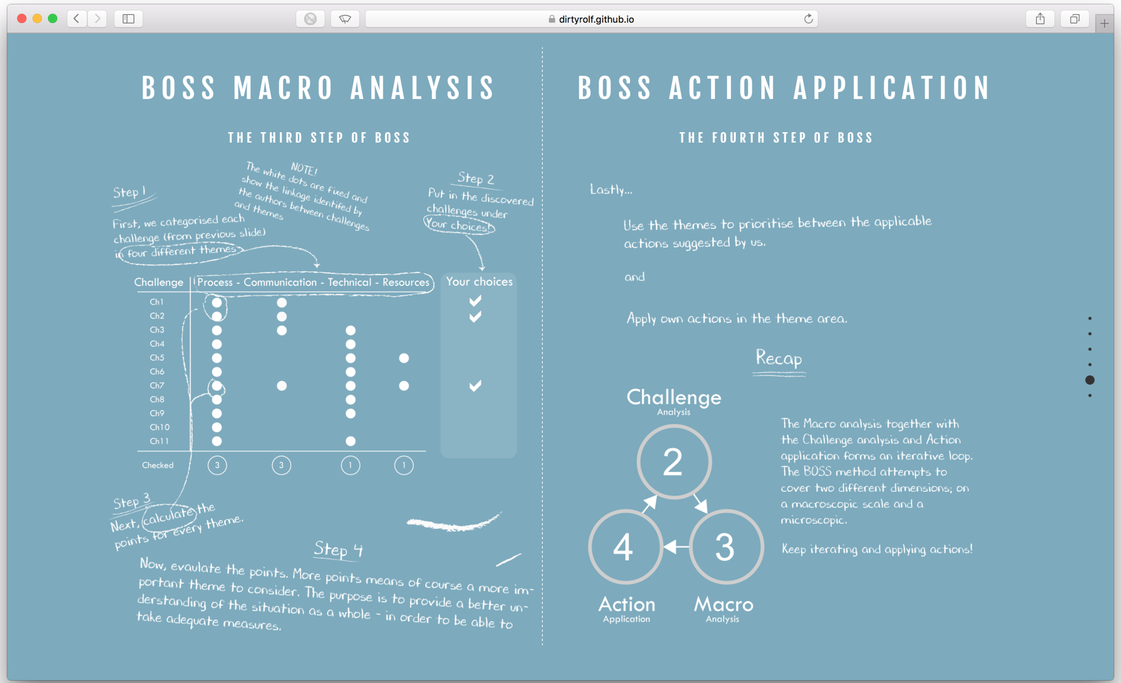
As a project grows, the need for formal specification grows with it and deciding how to allocate resources to the process is difficult.

Establish a requirements specification document after the first few rounds of a successful MVP

As the need for specifying requirements seem to be a function of growth, plan to hire a person that can be tasked with requirements management. Alternatively, re-allocate developer time.

🟥 Challenge 🟩 Potential action

Figure 10: BOSS Challenge Analysis with hover and click



BOSS MACRO ANALYSIS

THE THIRD STEP OF BOSS

Step 1

First, we categorised each challenge (from previous slide) in four different themes

NOTE!
The white dots are fixed and show the linkage identifed by the authors between challenges and themes

Step 2

Put in the discovered challenges under Your choices!

Challenge	Process	Communication	Technical	Resources	Your choices
Ch1	●	●			✓
Ch2	●	●			✓
Ch3	●	●	●		
Ch4	●		●		
Ch5	●		●	●	
Ch6	●		●		
Ch7	●	●	●	●	✓
Ch8	●		●		
Ch9	●		●		
Ch10	●				
Ch11	●		●		
Checked	3	3	1	1	

Step 3

Next, calculate the points for every theme.

Step 4

Now, evaulate the points. More points means of course a more important theme to consider. The purpose is to provide a better understanding of the situation as a whole - in order to be able to take adequate measures.

BOSS ACTION APPLICATION

THE FOURTH STEP OF BOSS

Lastly...

Use the themes to prioritise between the applicable actions suggested by us.

and

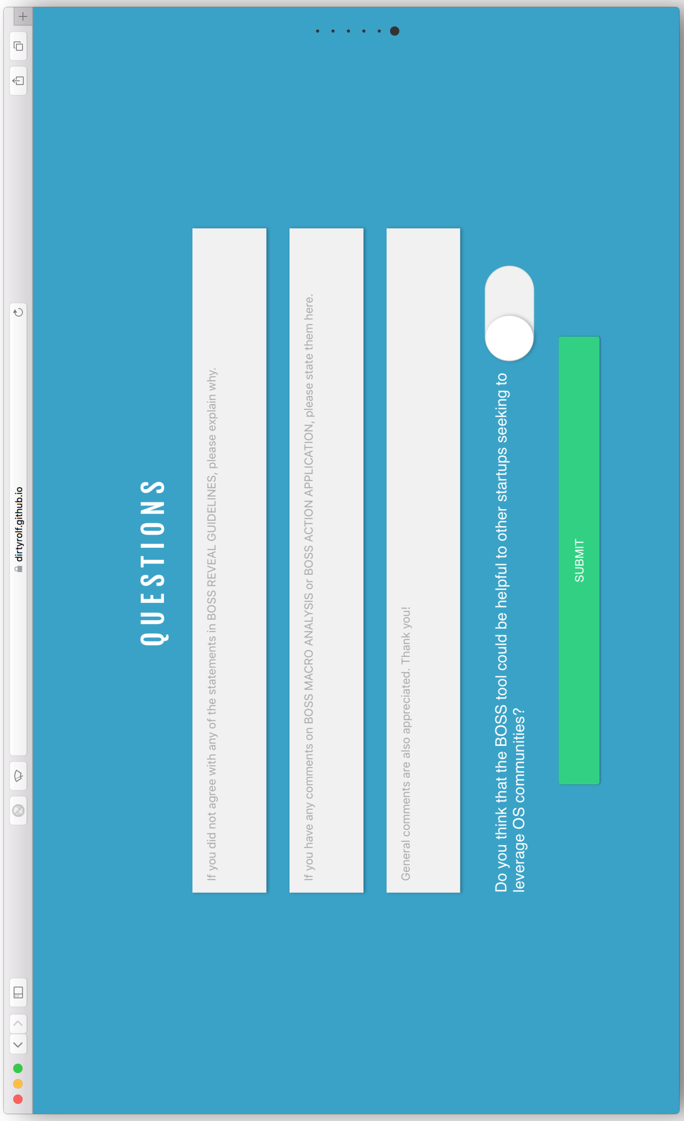
Apply own actions in the theme area.

Recap

The Macro analysis together with the Challenge analysis and Action application forms an iterative loop. The BOSS method attempts to cover two different dimensions; on a macroscopic scale and a microscopic.

Keep iterating and applying actions!

Figure 11: BOSS Macro Analysis and BOSS Action Application


dirtyrolf.github.io

QUESTIONS

If you did not agree with any of the statements in BOSS REVEAL GUIDELINES, please explain why.

If you have any comments on BOSS MACRO ANALYSIS or BOSS ACTION APPLICATION, please state them here.

General comments are also appreciated. Thank you!

Do you think that the BOSS tool could be helpful to other startups seeking to leverage OS communities?

SUBMIT

Figure 12: Questions

References

[1] Thomas Alspaugh, Walt Scacchi, and others. Ongoing software development without classical requirements. In *Requirements Engineering Conference (RE), 2013 21st IEEE International*, pages 165–174. IEEE, 2013.

[2] Steve Blank. Why the lean-startup changes everything. *Harvard Business Review*, May 2013.

[3] Andrea Bonaccorsi and Cristina Rossi. Why Open Source software can succeed. *Research Policy*, 32(7):1243–1258, July 2003.

[4] Alan Boulanger. Open-source versus proprietary software: Is one more reliable and secure than the other? *IBM Systems Journal*, 44(2):239–248, 2005.

[5] Henry W. Chesbrough and Melissa M. Appleyard. Open innovation and strategy. *California management review*, 50(1):57–76, 2007.

[6] Creative Commons. About the licenses. `https://creativecommons.org/licenses/`, 2016. Accessed 2016-05-23.

[7] Kevin Crowston, Hala Annabi, James Howison, and Chengetai Masango. Towards a portfolio of FLOSS project success measures. *Syracuse University Surface*, 2004.

[8] Joseph Feller and Brian Fitzgerald. A framework analysis of the open source software development paradigm. In *Proceedings of the twenty first international conference on Information systems*, pages 58–69. Association for Information Systems, 2000.

[9] Daniel M. German. The GNOME project: a case study of open source, global software development. *Software Process: Improvement and Practice*, 8(4):201–215, 2003.

[10] Carmine Giardino, Michael Unterkalmsteiner, Nicolo Paternoster, Tony Gorschek, and Pekka Abrahamsson. What do we know about software development in startups? *IEEE, Software*, 31(5):28–32, 2014.

[11] Joachim Henkel. Selective revealing in open innovation processes: The case of embedded Linux. *Research Policy*, 35(7):953–969, September 2006.

[12] GitHub Inc. Open source license usage on github.com. `https://github.com/blog/1964-open-source-license-usage-on-github-com`, 2016. Accessed 2016-05-23.

[13] Open Source Initiative. Open Source licenses standards. `https://opensource.org/licenses`, 2016. Accessed 2016-05-23.

[14] Terhi Kilamo, Imed Hammouda, Tommi Mikkonen, and Timo Aaltonen. From proprietary to open source—Growing an open source ecosystem. *Journal of Systems and Software*, 85(7):1467–1478, July 2012.

[15] Eriks Klotins, Michael Unterkalmsteiner, and Tony Gorschek. Software Engineering Knowledge Areas in Startup Companies: A Mapping Study. In João M. Fernandes, Ricardo J. Machado, and Krzysztof Wnuk, editors, *Software Business*, volume 210, pages 245–257. Springer International Publishing, Cham, 2015.

[16] Martin Michlmayr, Brian Fitzgerald, and Klaas-Jan Stol. Why and How Should Open Source Projects Adopt Time-Based Releases? *IEEE Software*, 32(2):55–63, April 2015.

[17] Kumiyo Nakakoji, Yasuhiro Yamamoto, Yoshiyuki Nishinaka, Kouichi Kishida, and Yunwen Ye. Evolution patterns of open-source software systems and communities. In *Proceedings of the international workshop on Principles of software evolution*, pages 76–85. ACM, 2002.

[18] John Noll. Requirements acquisition in open source development: Firefox 2.0. In *Open Source Development, Communities and Quality*, pages 69–79. Springer, 2008.

[19] John Noll, Sarah Beecham, and Dominik Seichter. A Qualitative Study of Open Source Software Development: The Open EMR Project. pages 30–39. IEEE, September 2011.

[20] John Noll and Wei-Ming Liu. Requirements elicitation in open source software development: a case study. In *Proceedings of the 3rd International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*, pages 35–40. ACM, 2010.

[21] Nicolò Paternoster, Carmine Giardino, Michael Unterkalmsteiner, Tony Gorschek, and Pekka Abrahamsson. Software development in startup companies: A systematic mapping study. *Information and Software Technology*, 56(10):1200–1218, October 2014.

[22] Eric S. Raymond and Thyrsus Enterprises. *The cathedral and the bazaar.* 2012.

[23] Per Runeson, Martin Höst, Austen Rainer, and Björn Regnell. *Case Study Research in Software Engineering.* Wiley, 2012.

[24] W. Scacchi. Understanding the requirements for developing open source software systems. *IEE Proceedings - Software*, 149(1):24, 2002.

[25] Matthias Stürmer and Thomas Myrach. Open source community building. *Licentiate, University of Bern*, 2005.

[26] Jose Teixeira, Gregorio Robles, and Jesús M. González-Barahona. Lessons learned from applying social network analysis on an industrial Free/Libre/Open Source Software ecosystem. *Journal of Internet Services and Applications*, 6(1), August 2015.

[27] Eric von Hippel. Innovation by user communities: Learning from open-source software. *MIT Sloan Management Review*, pages 82–86, 2001.

[28] Tobias Weiblen and Henry W. Chesbrough. Engaging with Startups to Enhance Corporate Innovation. *California Management Review*, 57(2):66–90, 2015.

[29] Joel West and Siobhán O'mahony. The Role of Participation Architecture in Growing Sponsored Open Source Communities. *Industry & Innovation*, 15(2):145–168, April 2008.

EXAMENSARBETE Requirements Engineering in Startups with Open Source Software related business strategies
STUDENT Billy Johansson, Martin Lichstam
HANDLEDARE Björn Regnell (LTH)
EXAMINATOR Martin Höst (LTH)

Leveraging volunteer communities for product development in new software companies.

POPULÄRVETENSKAPLIG SAMMANFATTNING **Billy Johansson, Martin Lichstam**

Small and innovative software companies, startups, are becoming increasingly more important. They strive to create ambitious solutions with a minimal team while facing constant resource constraints. In this thesis we examine how startups can leverage volunteer communities for product development.

Open Source Software (OSS) is software that is developed openly and available to anyone. Requirements Engineering (RE) is the notoriously difficult process of determining what a consumer actually wants from a product. By releasing software as OSS, a startup can gain access to a volunteer community that can contribute to the development of the software while simultaneously acting as consumers. We interview four different startups with ongoing OSS projects, examine how they manage the projects from an RE perspective, identify challenges related to the process and produce BOSS; a tool designed to guide a startup through an OSS venture.

Traditionally, the RE process has been a very formal and slow process with well defined tools for acquiring, specifying and validating requirements before releasing the product. In startups, a new methodology called LEAN has emerged. In the interest of quickly verifying a consumer's needs, the LEAN method advocates developing a Minimum Viable Product (MVP) showcasing the bare minimum of core features and getting it in the hands of consumers extremely early. After consumer feedback, features are adjusted and the process starts over. The OSS RE process is similarly very informal compared to traditional RE. However, we find that informality is not enough to make startups compatible with OSS, in fact there are a number of challenges related to merging the two.

Startups want to be able to change rapidly while OSS projects need to allow new developers to quickly understand the current state and intended future of the project. This results in very different RE processes. To further complicate matters, a startup can choose to release an entire product, or components of a product, as OSS. This brings challenges of identifying such components and applying different RE processes internally and towards the community.

In our thesis we produce a tool called Bridging Open Source and Startups (BOSS) to help startups leverage OSS communities for product development. The tool allows a startup to quickly identify components for an OSS release and iteratively guides the startup through the management process. This is done by identifying challenges we have extracted from interviews, diagnosing broader issues and applying potential actions suggested by us.