CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Inner Source In Product Projects

A Multiple Case Study Within A Company

Master's thesis in Software Engineering

Jakob Csörgei Gustavsson
Peter Eliasson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2017

Master's thesis 2017

Inner Source In Product Projects

A Multiple Case Study Within A Company

Jakob Csörgei Gustavsson
Peter Eliasson

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2017

Inner Source In Product Projects
A Multiple Case Study Within A Company
Jakob Csörgei Gustavsson & Peter Eliasson

Supervisor: Eric Knauss, Department of Computer Science and Engineering
Advisor: Maximilian Hristache & Sima Nordlund, Ericsson
Examiner: Regina Hebig, Department of Computer Science and Engineering

Master's Thesis 2017
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2017

Inner Source In Product Projects
A Multiple Case Study Within A Company
Jakob Csörgei Gustavsson & Peter Eliasson
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Background: Inner source is the concept of applying open source development practices in the context of a company. Reported benefits of using inner source include higher-quality software and more efficient reuse of software components. While the field of inner source research has explored the traits of inner source projects whose resulting software is to be used internally, research is lacking for inner source projects that have an external party as the end customer, which we call product projects.

Aim: In this thesis, we identify three especially interesting aspects of using inner source to develop product projects: feature prioritization, license issues related to third party products, and how to market the inner source project inside the company.

Method: We investigate how two large inner source projects at Ericsson handle these aspects of inner source by performing a multiple case study with semi-structured interviews as the primary method of data collection and thematic analysis for data analysis.

Results: Through analysis of the case study we identify key practices regarding the three aspects of inner source software development under study.

Conclusions: Based on the results in this thesis, we outline directions for future research on the topic of inner source in product projects.

Keywords: Case study, inner source, internal open source, customer-facing inner source, thesis.

Acknowledgements

We would like to express our thanks to everyone that was involved in this project. In particular, the following people for their help and support in making this thesis possible:

- *Eric Knauss, supervisor at Chalmers University of Technology:* For the guidance and academic expertise he has provided.
- *Maximilian Hristache & Sima Nordlund, supervisors at Ericsson:* For always being available for support and questions whenever it was needed.
- *The JCAT and UI SDK project members:* For always being happy to lend their valuable time for our interviews and follow-up questions.
- *Sebastian Blomberg & Joel Severin:* For being opponents and providing peer review feedback on multiple occasions.
- *Ulf Hansson:* For reviewing and giving feedback.

Contents

List of Figures

List of Tables

1

Introduction

In this thesis, we aim to further explore the topic of inner source in collaboration with a case company. In particular, the thesis aims to examine aspects of inner source that are seen as interesting when including external customers into an inner source project, an area where little previous research has been performed. The thesis was carried out at Ericsson, which was selected due to previously having shown interest in inner source (Torkar et al., 2011) and its large size (roughly 116000 (Ericsson, 2017)), making it more likely to find inner source projects there compared to a smaller company.

The remainder of this section starts by introducing the background leading up to the definition of the inner source concept. Following that is the problem statement in which we highlight the current lack of research regarding inner source projects used directly by a customer, and how we aim to fill this research gap. The problem statement is then condensed into three research questions, tailored to investigate aspects that should be taken into consideration when an external customer is involved. The thesis scope and limitations are then discussed, followed by a summary of the key results obtained in the study. Finally, the outline of the remainder of the report is presented.

1.1 Background

The successes of open source projects, such as the Linux Kernel, PHP, MySQL, Wordpress and Firefox, have in recent years opened the eyes of major software companies to the benefits of the open source model of software development. This can for example be illustrated by the fact that 85% of pull requests submitted to repositories on GitHub in 2016 were submitted to repositories run by an organization, a growth of over 760% from 2013 (GitHub, 2016). Open source software development is characterized by frequent releases, highly modular architectures, dynamic decision-making structures that can change based on fleeting requirements, and users being treated as co-developers (Robles, 2004). The last characteristic is essential to the open source way of development, in which users are encouraged to submit new features, bug fixes, file bug reports, write documentation, etc. Having users act as

developers can also result in access to a larger spread of expertise in the community, increasing the likelihood that there exists a developer in the project that is already familiar with any given issue. Additionally, the need for a modular architecture in order to facilitate contributions from newcomers further drives the technical quality of the software.

Several different authors have argued that lessons can be drawn from the development model used by open source communities and that it is possible to transfer best practices found in these communities to software development carried out in the context of a company (Asundi, 2001; Mockus et al., 2002; Fitzgerald, 2011); this is commonly referred to as inner source (Stol et al., 2014). Although inner source is a relatively novel concept that lacks comprehensive research (only 43 scientific publications have been published on the topic in 15 years, with the oldest dating back to 2002 (Capraro & Riehle, 2016)), a number of major software companies, such as HP (Dinkelacker et al., 2002; Melian, 2007), Philips (Wesselius, 2008; van der Linden, 2009; Lindman et al., 2010), Nokia (Lindman et al., 2013), IBM (Vitharana et al., 2010), Google (Whittaker et al., 2012), Microsoft (Asay, 2007) and Ericsson (Torkar et al., 2011), have found success through applying the concept. Capraro & Riehle (2016) identify a number of major benefits from implementing inner source such as cost and risk sharing across organizations, the ability to make better use of competences missing at component providers, faster time-to-market, and a wider openness and availability of knowledge.

1.2 Problem Statement

Although there is some literature in the inner source field, it is far from extensive. Indeed, in a recent literature survey performed by Capraro & Riehle (2016), the number of inner source-related publications was found to be 43 in total. In the same survey, Capraro & Riehle (2016) identify that most aspects of inner source are still in need of additional research in order to extend and validate current proposed ideas and frameworks. Furthermore, they find it unlikely that the field of inner source research can be seen as fully explored; instead, additional research is suggested in order to better establish the inner source field itself.

Most publications on the topic of inner source focus on internal inner source projects, which we define as inner source projects that are exclusively used internally. As this is the most common type of inner source project, we will henceforth use “internal inner source project” and “inner source project” interchangeably, and clearly note when another type of inner source project is intended. Examples of what can be regarded as internal inner source projects are found in literature describing Philips (Wesselius, 2008; van der Linden, 2009), Microsoft (Asay, 2007), Google (Whittaker et al., 2012), HP (Dinkelacker et al., 2002) and IBM (Vitharana et al., 2010). For example, the inner source project at Philips (van der Linden, 2009) is a shared product platform, which is then extended by multiple individual product teams

to fit the needs of their own product. The end products in this example involve some external customer, but the inner source project itself (i.e. the shared product platform) does not.

This same structure, as can be seen in the examples above, of isolating the inner source project from external customers seems to generally repeat itself in most literature on the area. It would, therefore, be interesting from an academic perspective to explore aspects of inner source projects related to the inclusion of external customers, projects which we will henceforth refer to as *product projects*. This interest was also expressed from an industry perspective by the case company studied in this thesis. In a context with an external customer, some issues might be regarded as of higher importance than if the project was developed for an internal customer. For example, usage of third party products might prove more problematic as certain licenses require making the product source code available to users of the product, which is naturally more sensitive if the user is external to the company.

Three aspects in particular were, in collaboration with the case company, identified as interesting when an external customer is involved: feature prioritization, a subset of legal issues regarding software licenses, and discoverability. The three aspects are further defined below:

- **Prioritization** pertains to how feature requests and feature submissions made to an inner source project are handled in practice. The term prioritization should be taken to mean any of the following:
  - how features are prioritized against each other during elicitation,
  - how the dedicated developers decide if they should take an elicited feature or if it should be delegated,
  - and how it is decided if an implemented feature should be accepted or not, based on need and alignment with the system as a whole.
- **Legal issues regarding software licenses** focuses on the notion that when contributions are made, the licenses of included third party dependencies have to be in accordance with the license of the final product. Making sure that no code is included from license-incompatible third parties is therefore essential for any commercial software.
- **Discoverability** is aimed at better understanding how an existing inner source project can spread awareness of itself within the organization, constituting the first step in the process of attracting new users and contributors to the project.

1.3 Research Questions

The following research questions have been chosen to explore the three aspects of inner source in product projects that were outlined in the problem statement:

RQ1. How are features prioritized in inner source projects?

RQ2. How are legal issues related to the usage of third party products handled in inner source projects?

RQ3. How can an inner source project spread awareness of its existence within the company where it is developed? (*Discoverability*)

The concept of third party products may in this thesis be assumed to be synonymous with third party software or third party libraries. The terms will henceforth be used interchangeably. The aspect of discoverability in RQ3 relates to how potentially interested employees of the company in question can become aware of the project's existence, not how to make employees interested in using or contributing to the project; the latter aspect is covered extensively in literature (Wesselius, 2008; Stol et al., 2014; Capraro & Riehle, 2016). The term discoverability will be used occasionally throughout the thesis to refer to RQ3.

1.4 Scope/Limitations

The legal issues considered for this thesis are focused exclusively on aspects regarding licenses, as we believed this to be the most common legal problem encountered when handling third party software. Other aspects, such as rules and practices pertaining to trade compliance, are such seen as out of scope and will not be discussed in the thesis.

1.5 Thesis Outline

The rest of this thesis is structured as follows:

- Section 2 provides a background on topics discussed throughout the thesis.
- Section 3 presents the used methodology and a motivation of why the employed methodology was chosen.
- Section 4 shows the results of the case study, in which interviews conducted with employees of the case projects have been analyzed. At the end of the results of each research question, a discussion is given. This section also provides

results of the validation workshop that was conducted.

- Section 5 discusses potential threats to validity and gives recommendations on future work.
- Section 6 summarizes and concludes the thesis.

2

Related Work

In this section, we provide additional background to topics relevant for the remainder of the thesis. The section starts by introducing open source, followed by an introduction to inner source. Finally, an inner source classification framework is presented.

2.1 Open Source

Open source software is software that comes with its source code attached, and with which the copyright holder has attached a license that may permit the studying, modifying, distributing and reselling of the software for any purpose (Perens, 1999).

Open source development practices differ significantly from those in the corporate world (Gurbani et al., 2006). One example of this difference is in the process of eliciting requirements. In corporate software development, requirements are found and decided on through the involvement of many disciplines such as marketing, business intelligence, research and development, management, and possibly more. Decisions are made based on business need and in a search for company profit. However, in open source projects, the users of the software are typically the developers themselves. Consequently, these projects rely on the developers to come up with requirements; since they are users, they themselves are expected to best know which features are needed and which bugs are the most critical to fix.

Another example in which open source software differs from corporate development is in its open and globally distributed development process, which increases coordination and communication (Fitzgerald, 2011). The secret behind the coordination model is in its structure of having a core team of expert developers who write the majority of the code. These developers are supplemented by a large number of additional coders and bug-fixers from the user population, increasing productivity without significantly impacting technical debt. When it comes to communication, the community of an open source project coordinates through informal means such as e-mail, forum posts, and version control systems.

Raymond (1999) shows a real-world example of successfully applying typical open

source development practices as he describes the development of the "fetchmail" client. In the same paper Raymond (1999) also establishes the cathedral and bazaar analogy, which is a recurring analogy within the open source and inner source fields. The cathedral in the analogy describes a company with a traditional top-down structure where only an exclusive group of software developers, such as the developers of a software product, have access to the source code. The bazaar, on the other hand, is Raymond's analogy for software developed over the internet, to which anyone can contribute. Raymond credits Linus Torvalds of Linux Kernel fame for inventing this process, and coins the expression Linus's law, which in its succinct form can be formulated as: "given enough eyeballs, all bugs are shallow". The intention behind this expression is that in a bazaar project such as the Linux Kernel, there are enough developers of different expertise that any found bug will have at least one community member finding the solution trivial. This aspect is also the core difference between the cathedral and bazaar styles of development. Cathedral projects employ groups of dedicated developers to hack away at tricky bugs and development problems until there is some confidence that they are all fixed, which leads, in Raymond's words, to long release intervals and eventual disappointment when it turns out the releases are not perfect. In contrast, bazaar projects release frequently, and subsequently "[...] [bugs] turn shallow pretty quickly when exposed to a thousand eager co-developers pounding on every single new release" (Raymond, 1999).

2.2 Inner Source

Recently, Wesselius (2008) used the same cathedral and bazaar analogy as Raymond (1999) when describing how internal software development processes at Philips Healthcare changed to incorporate certain open source practices. Wesselius likens the teams using these adopted practices to the bazaar, residing within the otherwise regular ways of working, seen as the cathedral. By encouraging the creation of an internal community that makes it possible for teams to share changes made to common assets, some of the same benefits as seen in open source projects can be observed also in the more closed company setting. The alteration of open source practices to be used within a company setting is what is referred to as inner source (Stol et al., 2014).

Despite the fact that inner source is based on the software development practices used in open source, the company aspect requires the open source practices to be tailored slightly in order to fit. Capraro & Riehle (2016) present a qualitative model that characterizes inner source based on four elements, extracted from surveying the inner source field:

- **Open environment:** Project artifacts such as code and documentation are open, external contributors are invited, and an open communication is established through the use of e.g. forums.

- **Shared cultural values:** Developers in inner source identify with the inner source projects they are involved in and the inner source community of those projects, rather than only the product or component they mainly work on. Furthermore, inner source projects typically welcome anyone who is willing to contribute and judge contributions meritocratically (Riehle et al., 2009).

- **Communities around software:** Communities, "an informal organization of individuals that communicate and collaborate with each other" (Capraro & Riehle, 2016), form around inner source projects and cross-organizational unit boundaries. The community is a key element of inner source.

- **Inner source development practices:** For example participatory reuse, which Capraro & Riehle (2016) define as developers contributing back to the software which they are reusing. Another example is the practice of different organizational units coming together to create inner source components which they have a shared interest in.

Riehle et al. (2009) suggest that some of the major benefits of implementing open source practices in an organization, i.e. implementing inner source, are as follow:

- **Volunteers and motivated contributors:** If a developer voluntarily joins a project, it likely means he or she is motivated to contribute. A highly motivated developer generally produces higher quality code than one who is not motivated.

- **Better quality through quasi-public internal scrutiny:** When all code in a project is visible to anyone in a large circle of people, there is more incentive to produce high-quality code as it can lead to positive effects on one's status.

- **Broad expertise:** As anyone can contribute to the project, this will lead to developers of different expertise being able to contribute. This can be stated through the previously mentioned Linus's law in its succinct form: "given enough eyeballs, all bugs are shallow" (Raymond 2001).

- **Broad support and buy-in:** Developers from across several development units will give rise to more support for the project from e.g. management.

- **Better research-to-product transfer:** Product developers can be integrated into research projects, which helps to bridge the gap between research and product development.

Common roles in an inner source project can be identified by examining case studies conducted in companies utilizing the inner source way of working. Projects that eventually become inner source projects are usually not started by management as a strategic project but are rather grassroots movements by individuals, teams or organizations (Melian, 2007). The person or persons who initiated the project then become the benevolent dictator (Gurbani et al., 2006), a term which is commonly found in open source projects (Mockus et al., 2002) such as the Linux kernel, where

this role is assumed by Linus Torvalds (Raymond, 1999). As open source projects often subscribe to the idea of meritocracy (Riehle et al., 2009) where the most able and experienced developers organically receive the most responsibility, contributors who become experts in an area of the project may be promoted to so-called trusted lieutenants (Gurbani et al., 2006). The benevolent dictator along with the trusted lieutenants form a core team who together steer the project. This structure is commonly seen in both open source projects and inner source projects.

One success factor of starting an inner source project and making it attractive to contributors and users alike is to have an initial product that solves a concrete problem, a so-called seed product (Stol et al., 2014). The idea that at least a basic implementation is needed before attempting to build a community was also noted by Raymond (1999), but in the context of open source projects:

> "It's fairly clear that one cannot code from the ground up in bazaar-style. One can test, debug and improve in bazaar-style, but it would be very hard to originate a project in bazaar mode. Linus [Torvalds] didn't try it. I didn't either. Your nascent developer community needs to have something runnable and testable to play with."

There are in principle two models of adopting inner source by an organization as observed by Gurbani et al. (2010): infrastructure-based inner source and project-specific inner source. In infrastructure-based inner source, the organization in which inner source is adopted provides critical infrastructure to facilitate inner source development such as web servers, mailing lists, code repositories, etc. An example of an infrastructure that can house inner source projects is GitHub (likely the enterprise version GitHub Enterprise in the context of an organization), or the similar software known as SourceForge which Gurbani et al. (2010) mentions as an example. Gurbani et al. (2010) suggests that the infrastructure-based inner source model may be successful when the candidate projects are primarily discrete software packages such as compilers, shells or utility applications. In project-specific inner source, the idea is that some organizational unit takes over a critical resource, or a shared asset, and makes it available to the larger audience. Gurbani et al. (2010) argues that this model is appropriate when the software is a primary technology of the company, along with perhaps being relatively immature and still evolving, and that the cost of redevelopment outweighs the cost of commonality. Stol et al. (2014) classified inner source programs at nine companies in accordance with these two models and found that seven of them were infrastructure-based and two were project-specific, suggesting that the infrastructure-based model is more widely used than the project-specific one.

The inner source concept is further expanded upon by Capraro & Riehle (2016), who introduce a distinction between *inner source projects* and *inner source programs*. The definition of an inner source program considers inner source at a larger organizational scale, encompassing the infrastructure-based and project-specific models previously mentioned. The inner source project concept regards individual projects. The definitions for inner source programs and inner source projects follow below,

where IS should be understood as an abbreviation for inner source:

> "An IS program is a coordinated effort of an organization to run and maintain one or multiple IS projects." (Capraro & Riehle, 2016)

> "An IS project is a software project with the goal to develop and maintain IS software." (Capraro & Riehle, 2016)

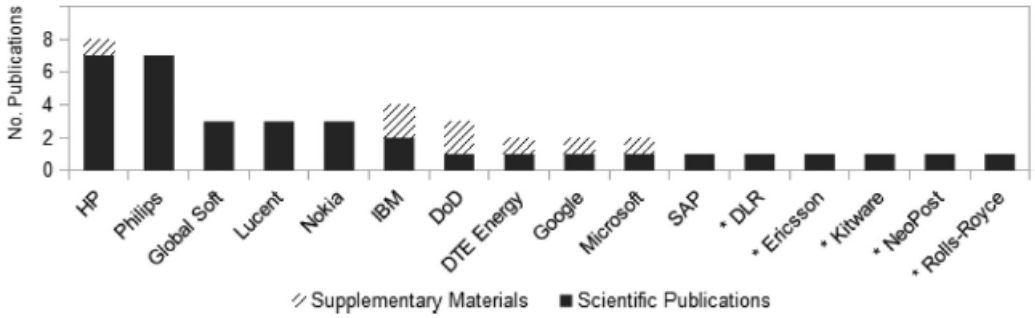

Figure 2.1: Inner source-related publications by organization. (Capraro & Riehle, 2016)

A number of major software companies have attempted to develop, or are developing, software using inner source, an assertion which is strengthened by the summary by Capraro & Riehle (2016), presented in Figure 2.1, which shows the available inner source publications that were conducted in collaboration with a company. As can be seen, many of the world's largest software companies are doing inner source, such as HP (Dinkelacker et al., 2002; Melian, 2007), Philips (Wesselius, 2008; van der Linden, 2009; Lindman et al., 2010), Nokia (Lindman et al., 2013), IBM (Vitharana et al., 2010), Google (Whittaker et al., 2012), Microsoft (Asay, 2007) and Ericsson (Torkar et al., 2011). At Google, for example, all source code is kept in a single repository to which all developers have read access, and are encouraged to contribute (Whittaker et al., 2012). In addition, developers at Google are allowed to allocate 20% of their time for projects outside of their immediate responsibility. This may lead to volunteering, which according to Riehle et al. (2009) may result in more motivated developers who in turn produce software of higher quality than developers who are not motivated.

Most literature in the inner source field discuss projects that are internal only. One exception is the study of the Forge.mil project hosted at the United States Department of Defense (Martin & Lippold, 2011). That project is described by Capraro & Riehle (2016) as both an inner source and a partner source project, the latter meaning an inner source project that also includes external partners and is therefore not exclusively internal. However, the Forge.mil project is never described in much detail. Instead, the focus of the paper is on discussing the cultural resistance to open source development practices and how the project still managed to successfully implement inner source.

2.3 Inner Source Classification Framework

The distinction between inner source programs and inner source projects, as described in the previous section, is expanded upon by Capraro & Riehle (2016) in their proposal of a framework for classifying inner source programs and inner source projects. As this study does not discuss the larger scope that is inner source programs, the remainder of this section will instead present the parts of the framework that are relevant for classifying inner source projects. However, it should still be noted that the framework can be used to classify inner source programs.

Governance	**Objective**
Single Organizational Unit	Exploration-oriented
Multiple Organizational Units	Utility-oriented
All Organizational Units	Service-oriented

Table 2.1: Classification framework for inner source projects. (Capraro & Riehle, 2016)

In order to classify an inner source project, the framework proposes two dimensions as shown in Table 2.1. The governance dimension regards who is responsible for the project from an organizational perspective, while the objective dimension describes what the aim of the project is, again as seen from the organization hosting the inner source project. Each of the dimensions further consists of three classes. The governance classes are synthesized from previous inner source research, whereas the objective dimension uses the same classes for inner source project objectives as found suitable for open source projects (Nakakoji et al., 2002).

The three classes of the governance dimension range from being controlled by a single unit to a fully distributed ownership: single organizational unit, multiple organizational units, and all organizational units. The governance classes are further defined as follows, where IS should be understood as an abbreviation for inner source:

> "**Single organizational unit:** The IS project is explicitly governed by one single organizational unit.
>
> **Multiple organizational units (governance board):** The IS project is governed by a board formed of multiple organizational units.
>
> **All organizational units:** The IS project is not governed by a select group of organizational units. The ISS [Inner Source Software] component is seen as a commodity. Governance and ownership is shared between all organizational units in the organization." (Capraro & Riehle, 2016)

The objective dimension is the aim (Capraro & Riehle, 2016), or the primary goal (Nakakoji et al., 2002) of the project. The three classes used for classifying the objective of an inner source project are the same as Nakakoji et al. (2002) suggest for

classifying the primary goal of open source projects: exploration-oriented, utility-oriented, and service-oriented. The definitions used for the inner source project framework are:

> "**Exploration-oriented:** The IS project aims to make innovation accessible to the whole program-wide IS community. Nakakoji et al. [2002] note that due to their 'epistemic nature' such projects usually have high-quality requirements. Contribution of feedback (e.g., via mailing lists) is particularly important for an exploration-oriented project.
>
> **Utility-oriented:** The IS project aims to fill an immediate need in functionality. Typically, the developers of the initial code are an individual or a small party who 'cannot find an existing program that fulfills their needs completely' [Nakakoji et al. 2002]. Utility-oriented projects usually have only a small project-specific community or their community exists as part of a larger community (e.g., if the utility-oriented project is part of the ecosystem of another IS project).
>
> **Service-oriented:** The IS project's main goal is to provide 'stable and robust services' to end-users of the ISS software [Nakakoji et al. 2002]. Service-oriented projects typically produce business critical ISS software components, have high quality requirements, and are conservative against rapid changes [Nakakoji et al. 2002]." (Capraro & Riehle, 2016)

The above definitions of the objective dimension are somewhat ambiguous in that the goal of a project could fulfill multiple of these definitions. Especially the definitions for service-oriented and exploration-oriented projects above may seem very similar to each other; they both even contain the same wording of high-quality requirements. However, what is perhaps not evident in the quoted definitions above, as given by Capraro & Riehle (2016), is that the original definitions by Nakakoji et al. (2002) also include differences in how strictly community contributions are controlled. In an exploration-oriented project, code contributions are rarely accepted from the community; instead, the project is primarily developed by a small number of expert programmers in a manner similar to the cathedral phase described by Raymond (1999). In a service-oriented project, code contributions are more likely to be incorporated into the project. Still, a service-oriented project does not fully embrace a bazaar-style of development as such rapid changes could potentially make it difficult to provide a stable service. Finally, the utility-oriented project does often encourage a bazaar style of development, meaning contributions are usually accepted more liberally than the other two (Nakakoji et al., 2002).

3

Methodology

The method employed for this study was divided into a pre-study, a primary multiple case study, and a workshop to validate the findings from the primary study. The pre-study was conducted together with the case company to identify shared goals for the outcome of the study, in order to ensure that the study would be beneficial to both the case company and the academic field. The pre-study was also used to guide the design of the remaining research.

The structure of this section mirrors the structure of the study. First, Ericsson is introduced as the case company for the study. Following that is a section on the pre-study. A motivation for the primary study method is then presented, followed by a section describing the case study. Finally, the approach taken for the workshop that was held to help validate the findings is described.

3.1 Case Company: Ericsson

The case company selected for the study was Ericsson. Ericsson is among the largest ICT (Information and Communications Technology) companies in the world, ranking at 23rd based on revenue (Fortune, 2016), and employing more than 116000 people in many locations across the globe (Ericsson, 2017). The study was carried out at Ericsson's office in Gothenburg in Sweden.

A strong reason for selecting Ericsson as the case company for the study was that Ericsson had already shown interest in using inner source for software development. For example, this interest can be seen in their involvement in existing inner source-related research (Torkar et al., 2011), and later reinforced in discussions both before the study was started and during the early parts of the pre-study, as presented in the next section. Additionally, the benefits of inner source are arguably more interesting for companies of a fairly large size, as it likely increases the chances of enough people gathering around a project to form a community.

Ericsson was also seen as an appropriate case company due to its similarities with other companies at which inner source studies had been previously conducted. These similarities were expected to help to mitigate the risk that the related research might

not be applicable in the studied context. For example, Ericsson is, like many other companies that have adopted inner source such as Microsoft, IBM, and Philips, a globally distributed company that is focused on software and with a large number of employees (Ericsson 116000, Microsoft 121000, IBM 380000, Philips 46000) (Wikipedia, 2017).

3.2 Pre-Study

A pre-study was conducted due to some uncertainties in the expected outcome of the primary study, how the outcome would fit with the current inner source research, and to investigate the current state of inner source adoption at Ericsson. The goal of the pre-study was to answer these uncertainties and to guide the remaining research in a direction that would prove beneficial to Ericsson and the inner source research field. Additionally, the pre-study provided opportunities both for getting to know and for practicing using research methods that would later be used in the primary study, meaning some potential validity concerns could be addressed before the primary study started.

The first activity of the pre-study was a workshop held with four Ericsson employees (two line managers, one developer, one system test architect) and our academic supervisor. The workshop had little formal structure and instead focused on an exploration of the topic of inner source at Ericsson. The workshop was not transcribed, but notes and some documents were collected throughout the session. The primary outcome of the workshop was the realization that Ericsson was already running multiple inner source projects. In addition, the interest of knowing differences between product projects and internal inner source projects was brought up, which guided the formation of the three research questions.

Other than the workshop, multiple informal meetings were held to broaden the understanding of Ericsson's inner source initiative and to help in identifying what group of people would be suitable for further interviews. The people involved in this part were mostly those related to the organization within Ericsson for which this thesis was carried out. Two interviews were also conducted with employees associated with one of the inner source projects that were identified in the workshop. Although an interview guide was prepared for these interviews, it did not match the guide used later in the study, and as such these interviews were not included as data for the primary study.

3.3 Methodology Motivation

When selecting a research method, there are primarily two paradigms to take into account: methods relying on quantitative data and methods relying on qualitative

data (Runeson & Höst, 2009). Runeson & Höst (2009) summarize quantitative data as "numbers and classes", while qualitative data is described as "words, descriptions, pictures, diagrams etc". The objective of the research method can be categorized as exploratory, descriptive, explanatory and improving. The description of these four objectives is presented by Runeson & Höst (2009) as follows, interpreted from Robson (2002):

> "**Exploratory** — finding out what is happening, seeking new insights and generating ideas and hypotheses for new research.
>
> **Descriptive** — portraying a situation or phenomenon.
>
> **Explanatory** — seeking an explanation of a situation or a problem, mostly but not necessary in the form of a causal relationship.
>
> **Improving** — trying to improve a certain aspect of the studied phenomenon." (Runeson & Höst, 2009)

As our study primarily had an explorative objective, the choice of methodology was directed towards either a case study or a survey; although the latter is descriptive, it could also have fit our goals of trying to find out the state of things. However, because several aspects of inner source product projects were unknown at the outset, a survey was ultimately unfit as the questions asked in a survey should not be changed during the course of a study. In contrast, a case study may be changed and improved upon as more information emerges and more knowledge about the subject is obtained. For these reasons, case study was selected as the research method.

The performed case study was a multiple case study, where each project constituted a separate case in the study. By clearly separating each project into its own case, we believed that the cases would be more concrete, making the findings in each case easier to generalize as their contexts would be more clearly defined. Indeed by regarding each project separately, we believed that it would be possible to better describe and generalize the findings of each case by relying on the classification framework for inner source projects proposed by Capraro & Riehle (2016). In addition, we thought the multiple case study approach would better optimize time spent when not all teams could be interviewed during the same time period, which was deemed fairly important due to indications from the pre-study that some teams might be harder to reach. A multiple case study allows for more flexibility in this sense as the analysis of each case can be done more separately than if all projects were seen as part of the same case.

3.4 Multiple Case Study

This section describes the multiple case study that was performed in order to answer the research questions by investigating existing inner source projects at Ericsson.

The section starts by introducing the selected case projects (JCAT and UI SDK) in order to establish a context for the reader. Following that is a section on semi-structured interviews, representing the primary data collection method used for the case study. The thematic analysis method is then presented both generally, as well as how it was implemented in this study to analyze the interview data. Finally, the project classification section describes how the cases were classified according to the classification framework for inner source projects (Capraro & Riehle, 2016).

3.4.1 Cases

The selected cases for the case study were two projects within Ericsson that had already adopted an inner source way of working. These projects were discovered during the pre-study, where it was noted that no global listing of inner source projects seemed to exist. As such, inner source projects were picked that were known by our contacts at the company, rather than a more careful selection of cases. Both projects were exclusively internal, i.e. they did not have any external customers but rather produced a framework or a library that was used in other products. The cases are further described in this section, aiming to provide the reader with a better understanding of the context where these projects operated, as well as a short overview of what the projects are about.

Java Common Auto Tester (JCAT)

JCAT is a Java-based test automation framework developed for use within Ericsson. It is based on the open source frameworks JUnit and TestNG, extending these frameworks with the additional functionality needed in the testing of Ericsson products. It is currently the inner source project in Ericsson with the largest community, with over 10,000 unique active users. JCAT provides functionality to interact with products and technologies inside Ericsson in order to write automated test cases for these products. The framework also makes it possible to subscribe to both synchronous and asynchronous responses (such as alarms, alerts, and notifications) from the products via its interfaces. JCAT is independent of any product development organization. The idea is that organizations within Ericsson that develop a product use the JCAT framework and write their own extensions to it, further enabling testing for their own specific product.

Attached to JCAT (but also independent of the same) is a set of so-called common libraries. These are Java-based libraries that provide some useful set of functionality and may or may not be a plugin to JCAT itself. Common libraries leverage the JCAT community but are not an integral part of the framework.

User Interface Software Development Kit (UI SDK)

The UI SDK is a framework for building JavaScript web applications and web UIs. The project is split into three sub-projects: the Client SDK, the REST SDK and the Help SDK. The Client SDK is the primary and largest part of the project and consists of a large collection of components used for web UI and web application development, with each component split into a separate module hosted in its own code repository. The REST SDK details guidelines for implementing REST interfaces and the Help SDK serves as a rendering engine for online documentation.

The UI SDK was initially created for a product called Ericsson Network Manager, which at the time was being developed within the product development unit (PDU) "NAM". As the product matured and was later released, the UI SDK transitioned into an inner source project that encouraged contributions also from outside the PDU. However, the UI SDK and its core team are still strongly associated with the original PDU NAM, in that it is this unit that employs the core team.

3.4.2 Data Collection: Semi-structured Interviews

ID	Project	Experience In Project	Title In Project
J1	JCAT	5 years	Architect
J2	JCAT	1 year	Senior Developer
J3	JCAT	1 year	Senior Developer
J4	JCAT	5 years	Lead Developer
J5	JCAT	10 years	Upper Manager
U1	UI SDK	4 year	Upper Manager
U2	UI SDK	3 years	Senior Developer
U3	UI SDK	8 years	Architect
U4	UI SDK	4 years	Senior Developer

Table 3.1: People interviewed for the study, their experience, and their title in the project.

As the primary part of the data collection for the case study, it was decided to use semi-structured interviews. The decision to use semi-structured interviews was based on the need for a data collection method that was primarily exploratory, as to align with the goal of the overarching study. In addition, many key people were known to be located in offices in different countries, which made it infeasible to physically meet them. Techniques like observations, which require a physical presence, were therefore deemed inappropriate. Interviews, on the other hand, are generally easier to conduct in such a setting as they can just as well be performed over communication tools like Skype.

The first step of the data collection was to determine how many, and who, to interview in each project. The number of people to interview was decided on a flexible project-by-project basis and ended up being slightly different between the projects. In practice, new interviews were scheduled until it was seen that most points brought up in an interview had already been brought up in a previous interview. This was doable as interviews for each project were performed within a fairly narrow time span, making it possible to maintain a good memory of what had been said in earlier interviews. It was further believed that the role had a greater impact than the number of people interviewed, as most people in the project might simply not be aware of the issues investigated. The interview subjects were as far as possible chosen to be similar between the different cases, while also covering multiple roles. Table 3.1 summarizes the people that were interviewed for the study. The table does not include names of the people interviewed as to preserve anonymity. Some titles were also slightly altered, again to protect the anonymity of the people involved in the study.

The interviews were conducted according to general guidelines for semi-structured interviews given by several authors in the field (Hove & Anda, 2005; Runeson & Höst, 2009; Seaman, 1999; Taylor et al., 2016). An interview guide was prepared for use in the interviews (see Appendix A), both in English and in Swedish as some interview subjects were more familiar using the latter language. Before each interview, the interview subjects were informed of their absolute anonymity and the confidentiality of the information they provided. This is both in line with ethical guidelines for research and likely makes the subject more confident in being able to speak freely without fear of repercussions (Seale et al., 2004). As part of the pre-interview process, the intent of the research was also explained to the subjects. Audio for each interview was recorded with explicit consent from the interviewee and transcribed fully. In many cases, it was possible to obtain a video recording of the interview, which helped better the understanding of the context that might otherwise get lost in an audio-only recording. Additionally, notes were taken during the interviews to serve as quick reminders of what had been brought up by interviewees.

All interviews were conducted by both authors together, as using two interviewers provides a number of benefits (Hove & Anda, 2005). First, it makes the subjects talk much more. Hove & Anda (2005) performed an experiment where they interviewed the same subjects with both one and two interviewers and compared the results. It turned out that the interviewees talked on average 59% more when two interviewers conducted the interview, suggesting that more information could be retrieved by using two interviewers. Second, two interviewers will in most situations ask more questions than one would, simply as two people have more brain processing power than one. This can be used for coming up with follow-up questions on what the subject says, again leading to more information being collected. Third, it is often easier to conduct interviews since interview roles can be divided. For example, one interviewer primarily asks the questions while the other one listens intently to what is being said and attempts to come up with follow-up questions. This division works well because it is difficult to both listen to answers and at the same time plan for how to ask the next question in the interview guide. Lastly, two interviewers may

discuss their interpretations of what was said and come to new insights that else could not have been found.

Since both the authors of this study were inexperienced in interviewing, a technique for iteratively improving interviewing technique as described by Taylor et al. (2016) was used, in which interview recordings were transcribed shortly after the interviews had happened, especially for the initial ones. This has the effect of making oneself aware of how one conducts interviews and where questions may be unclear. For example, it was found early on that it might be necessary to explicitly ask for both title and role, rather than only the title, in the cases where an answer to both was interesting. Furthermore, transcribing allows the interviewer to both clearly see, and enable reflection over, where more probing of the subject's answers should have happened and more generally how the interviewer's skill could be improved.

3.4.3 Data Analysis: Thematic Analyis

To extract meaningful results from interview transcripts, it is necessary to perform an analysis of the data. How such an analysis is best performed depends on multiple factors. For example, the data analysis technique differs when performing an analysis of quantitative data as compared to analysis of qualitative data (Runeson & Höst, 2009). For this study, the collected data was purely qualitative since it only consisted of answers to open or semi-open interview questions. The objective of the data analysis was to extract new knowledge and to explore the general areas surrounding the research questions for the study. For analyzing data with these preconditions, the general method consists of first coding the data and then to arrange the codes into more general themes, following a method called thematic analysis (Runeson & Höst, 2009).

The thematic analysis performed in this study was performed as to closely follow the clearly defined process proposed by Braun & Clarke (2006). Although the paper focuses on the area the area of psychology, it was seen that the process would transfer fairly well also to other fields. By following a well-defined process, it is also possible to mitigate one of the perhaps most common criticisms against the method, namely that thematic analysis is sometimes seen as a method without any specified process (Braun & Clarke, 2006).

The remainder of this section is structured following the six phases suggested in the aforementioned thematic analysis paper, though we decided against including the last phase ("producing the report"). Further, we also decided to include a "prerequisite" phase. It should also be emphasized that thematic analysis is not generally performed linearly from the first to the last phase, instead it is often the case that one moves back to a previous phase multiple times throughout the process (Braun & Clarke, 2006).

Prerequisites Phase

As a prerequisite to performing thematic analysis, Braun & Clarke (2006) propose that the data should be categorized into two sets: data set and data corpus. The data set is the data used for the analysis, while the data corpus encompasses the entirety of the collected data. For this study, the data set that was used for the analysis was the transcripts created from the semi-structured interviews. The data set was chosen to only include interviews for multiple reasons, but primarily because audio recordings were available for the interviews, minimizing the risk of researchers interpreting the data already before the analysis. Additionally, not all people that took part of meetings or presentations were explicitly asked by the researchers to take part in the study. As such, including them could have been problematic from an ethical standpoint. The data corpus included the data set and extended it with some additional material, such as the documentation collected for the project classification described in a later section.

Phase 1 - Familiarizing Yourself With Your Data

To perform an analysis it is important to be familiar with the collected data set, which for a data set of interview transcripts means reading and rereading the transcripts (Braun & Clarke, 2006). We did this by going through the transcripts one at a time for each case. Every transcript was read by both of us in order to better remember what was said in that specific interview. This process was helped by the fact that both researchers had been present for each interview, meaning that transcripts were already familiar to the researchers, as well as the fact that transcribing had been performed by the researchers themselves.

Phase 2 - Generating Initial Codes

After having familiarized ourselves with the data set, each transcript was coded in a computer-based tool for qualitative data analysis called NVivo. The coding process was performed on a single computer with both researchers present, as to enable a better shared understanding of which code or codes each line corresponded to. Codes were initially kept rather precise in order to not prematurely filter out any relevant information. For example, codes were created both for “contribution guidelines” and “code guidelines”, despite these concepts being highly related. As more transcripts were coded it became clear that some codes could be removed or merged with similar ones, while others still would require splitting old codes to fully express the intention. Furthermore, some codes could already be identified as likely being related. In those cases, the codes were added to a shared parent node, forming a tree-shaped hierarchy. This entire process was performed in an iterative manner in that reflection on which codes were appropriate happened continuously throughout the coding process. In total, about 80 codes were created, visualized as a hierarchy

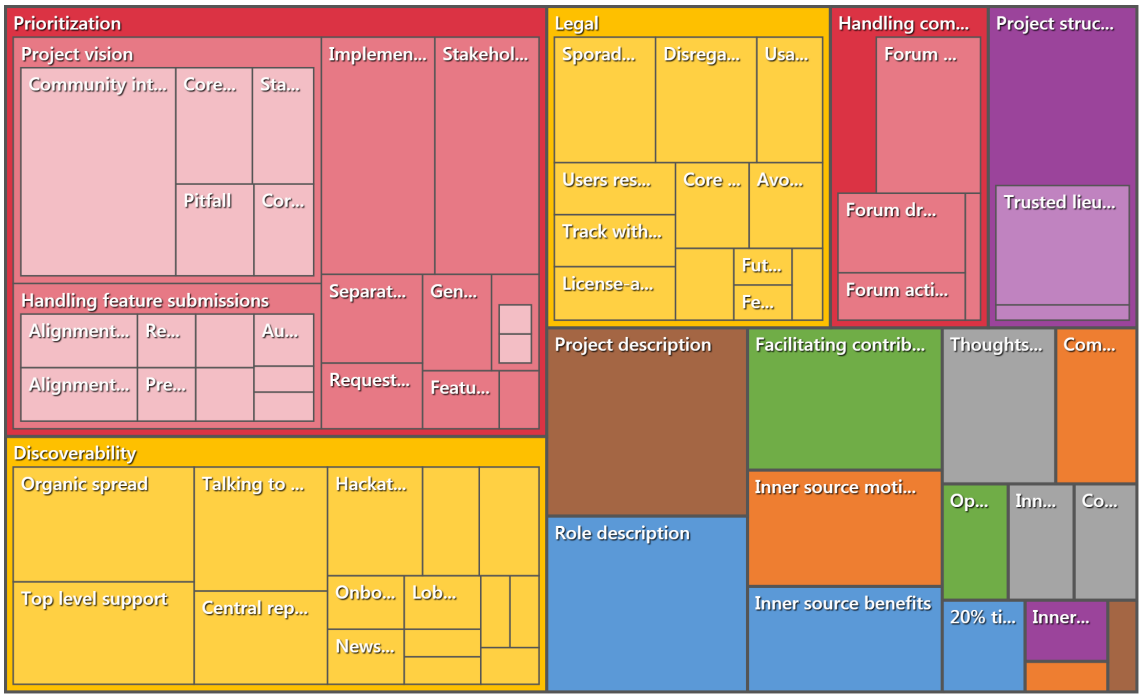

Figure 3.1: A hierarchy chart of codes extracted from the transcripts, created using the NVivo tool. Each box represents a code, and nested boxes indicate codes that are part of a hierarchical tree structure of codes. For example, the "Project vision" box (top left) is part of the "Prioritization" box, indicating that the "Project vision" code was placed under the "Prioritization" code.

chart in Figure 3.1.

Phase 3 - Searching For Themes

Figure 3.2: Codes written on post-it notes are grouped into potential themes on a whiteboard, allowing easier iteration and overview.

The initial codes were, after all transcripts had been coded, organized and merged to form overarching themes that should describe what the codes put into each theme were about. This step is where the meat of the data analysis starts, i.e. where the long list of codes, and the data they contain, are combined to tell a story of the data and to find relationships between the themes themselves. In this step, it is important to not discard anything, as it is still impossible to know which codes are the most relevant and which are not (Braun & Clarke, 2006). Some codes did not end up fitting into any of the found themes; these codes were instead placed in an uncategorized theme named “Miscellaneous”. This organization of codes was done by writing each code on a post-it note and then placing them on a whiteboard. This allowed for a good overview of all codes, and an easy way to iterate on what themes were most appropriate. Figure 3.2 illustrates this process, where themes of post-it notes are grouped and circled by a whiteboard marker. The candidate themes were then also transferred to the NVivo tool.

Phase 4 - Reviewing Themes

The next step of the process is to review and refine the potential themes found in the previous step. Some of the themes were found to, in fact, not be backed by enough data points or too much contradictory data, and were subsequently discarded, while some of the themes were merged to even broader themes on account of being too closely related. This review step consists of two distinct subphases conducted at different levels of data. The first level that needs to be considered is the actual transcript extracts that form the codes of the themes. It is important that these extracts are coherent and tell roughly the same story that the theme is supposed to signify. If they do not, it needs to be considered if the theme should be discarded, split into separate themes, or if the codes that contain incoherent extracts should be moved to any of the other candidate themes. The second level of consideration is whether the themes work in relation to the whole data set and in relation to each other, and how well the themes seem to represent the data set. As part of this level, it is necessary to re-read the entire data set, both to verify that the data matches up well with the candidate themes and to code any data that was missed during previous steps. The first time this phase was reached it was evident that some of the themes did not fit in with the larger data set. This prompted a revisit to the previous phase and a definition of new themes that would better fit the entirety of the data set.

Phase 5 - Defining And Naming Themes

The final phase (again, we disregard the "producing the report"-phase) of the thematic analysis process is to define and name the themes, which at this point were fairly stable. In this step the most important consideration is whether the themes and their underlying data fit the overall story that is being told about the data set, both in relation to the data set itself but also in relation to the other themes, in order to ensure that there is not too much overlap between the themes (Braun & Clarke, 2006). At the end of this step, one should be able to describe the scope of each individual theme in a few sentences at most. If this is possible, the themes are likely well-defined and of the correct size. This is also where the themes are given their final names, although working names will have already been established in earlier steps. A smaller presentation was held at the case company during this final phase, which provided a good feedback opportunity on the finalized themes.

3.4.4 Project Classification

As a final part of the case study, the cases were also classified according to the framework for inner source projects put forth by Capraro & Riehle (2016), described in the 2.3 Inner Source Classification Framework section. Although the results from the classification are not directly related to the research questions posed for the

study, classifying the cases was done in an effort to provide additional context for the reader. By knowing the types of projects that were part of the study, one can hopefully better judge the applicability of the study on other inner source projects. As defined by Capraro & Riehle (2016), the two dimensions that the projects were classified on were project governance and project objective.

In order to perform the classification, data was primarily set to be collected from passive data sources, such as documentation available for the projects. This was due to a strong belief that such information would indeed be obtainable from the available documentation, and as such the valuable interview time was better spent on the primary research questions. However, it was quickly discovered that interviewees often brought up information that could be used for the classification, even without any explicit questions being asked for this purpose. As interview data was seen as likely being more current than the documentation, it was used whenever possible, while the documentation was used in cases where either the data differed or there was not enough data. As the classification was seen as mostly separate from the primary results of the study, it was conducted in a slightly less methodical fashion. Pre-defined search strings or databases were not used; instead the documentation and interview transcripts were actively scanned to collect the necessary information. After having classified the projects, the classifications were validated with at least one team member of each project to make sure interpretations had been made correctly. This validation was performed by email, where the framework definitions were presented together with our classification (see 4.1 Project Classification).

It is worth noting that no classification was attempted for the overarching inner source program at Ericsson. Although a rough estimate built on the data from the cases studied might have been possible to create, it was decided that a classification of the inner source program would require a larger picture than could be gained from the projects that were investigated for this study.

3.5 Validation Workshop

As a final step of the study, a workshop was held together with Ericsson employees in order to both validate the findings of the primary study and to provide discussion opportunity for applying the findings to another project, called AAT, which plans to adopt inner source in the future. The AAT project aims to provide an automated acceptance test framework to be used by both internal and external users. In order to facilitate such discussion, the workshop was structured in two parts. The first part involved presenting the findings of the case study, the current state of inner source adoption of the AAT project, as well as more technical concerns of inner source project creation within Ericsson. The second part involved a brainstorming session on questions raised during the first part of the workshop, which was a largely unmoderated and involved a multitude of topics ranging from legal considerations to architectural details of the AAT project. The workshop lasted for about one and

a half hours.

Shared between all the participants was an interest in inner source, as well as some connection to the AAT project. The following people took part of the workshop:

- Three employees from an internal DevOps organization experienced in inner source implementation on the more technical level (i.e. creation of repositories, firewall configurations, etc.) as well as legal aspects of third party product handling.
- Multiple members of the AAT project: the project manager, two line managers, an architect and an engineer.
- A line manager related to the AAT project but not part of it.
- An engineer that had previously worked with AAT.

The workshop was not transcribed; instead, documented by having both researchers take notes of what was brought up. The notes were then discussed and combined shortly afterward to get a shared understanding of what had been said. As such, the results will be provided as a summary of three relevant areas discussed. No verbatim quotes will be reproduced. As the workshop was primarily carried out as validation of the case study, the discussion will mostly be focused on highlighting similarities and differences between the findings in the workshop and what was found in the case study.

4

Results

Research Question	Themes
RQ1 - How are features prioritized in inner source projects?	Step 1: Elicitation Step 2: Delegation of Implementation Step 3a: Core Team Implements Step 3b: Community Implements
RQ2 - How are license issues related to the usage of third party products handled in inner source projects?	Inner Source Project Implications • License Less Important • Users Responsible Using Third Party Products • Avoid Third Party Products • License-aware Usage of Third Party Products
RQ3 - How can an inner source project spread awareness of its existence within the company where it is developed?	Project Finds People People Find Projects Top-level Support Inner Source Program

Table 4.1: Overview of the identified themes by research question.

In this section, we present the findings from the analysis of the data collected throughout the study. First, related to the research questions is a section on project classification, in which the result of the inner source project classification is presented. After the classification section follows results for the three research questions, each question discussed separately. The section for each research question begins with a summary of the findings, followed by the found themes (see 3.4.3 Data Analysis: Thematic Analyis) in more detail with supporting quotes. At the end of each research question follows a discussion which highlights the main findings, interpretations of the findings, and evaluation of the findings in the light of related work. The final section presents the findings of the validation workshop. Table 4.1 provides an overview of the identified themes for each research question.

Project	Governance	Objective
JCAT	All Organizational Units	Service-Oriented
UI SDK	Single Organizational Unit	Service-Oriented

Table 4.2: Classification of the case projects along governance and objective dimensions, following an inner source project classification framework by Capraro & Riehle (2016).

4.1 Project Classification

The case projects were classified in accordance with the inner source classification framework previously described in 2.3 Inner Source Classification Framework. As presented in Table 4.2, the JCAT project is seen as having an all organizational units governance whereas the UI SDK can be observed to be governed by a single organizational unit. Both case projects have a service-oriented objective. The remainder of this section provides the relevant data that was used in classifying the projects.

4.1.1 JCAT

In this section, the classifications for the JCAT project is presented. It can be seen that the JCAT project is governed by all organizational units (i.e. there exists a type of collective ownership involving all stakeholders), primarily from how the project is funded. The objective is classified as service-oriented, mostly due to the bazaar-like handling of contributions and a strong commitment to backward compatibility.

Governance

The JCAT project is not explicitly owned by any single unit within Ericsson, but rather owned and sponsored by the company as a whole. As such, JCAT is classified as governed by all organizational units. This governance situation is explained in the following quote where JCAT is said to be part of an Ericsson global tool list, meaning that the sponsorship of the project is not tied to any specific organizational unit or units:

> "What helps here also is that there is a thing called global tool list. Which means that these are tools that are centrally sponsored from the R&D budget. There are only two test automation frameworks on this. And of course JCAT is one of them." (J5)

The same governance explanation was also found on the JCAT internal wiki page, stating again that JCAT is an Ericsson global tool:

> "JCAT is an Ericsson global tool, so Ericsson makes sure that there are sufficient resources assigned for development, maintenance and support activities, so JCAT will not be abandoned and we have guarantees for resolving bugs and implementing feature requests." (Ericsson internal JCAT wiki)

To clarify the ownership, the same person as above (J5) was asked if JCAT had a single owner:

> "No. Okay, it is a little bit 'fluffy' what is called unit, especially nowadays. You know the organization is just changing. Before there were BUs (Business Units), and now it's going to be business areas. But I still consider this as the highest possible unit level, directly under Börje [Ekholm, CEO of Ericsson]. Then from each of these current business units, and future business areas, we have users." (J5)

This can be understood to mean that JCAT is indeed not owned by any single unit or group of units. Instead, the ownership of JCAT is shared across all units within Ericsson.

Objective

The objective of the JCAT framework is classified as service-oriented, which can primarily be seen in how contributions are handled, but also in the commitment to backward compatibility. Outside contributions are both welcomed and encouraged, but the JCAT core team thoroughly validates each such contribution before it is accepted into the project:

> "[...] we, the architecture team, both control the quality of submissions to make sure that they don't break the architecture, and also provide architecture suggestions when a contributing user comes and asks 'We would like to implement support for this, is that a good idea?' " (J1, translated from Swedish)

The project's welcoming stance on contributions is further emphasized by another team member, again with the condition that contributions can be verified to be of sufficiently high quality:

> "[...] if someone sees that they would like to contribute something back to the community, then they're free to go. It's just going to go through the review process to secure quality and backward compatibility and such." (J5)

The above quote also mentions backward compatibility, another factor fitting with the high-quality requirements of a service-oriented project.

4.1.2 UI SDK

The UI SDK is classified as governed by a single organizational unit, as it can be clearly seen to be owned by one specific development unit. The UI SDK project has a service-oriented objective as it provides a high-quality framework with a strong backward compatibility focus, while also fitting with the controlled bazaar-style of contributions that is common for service-oriented projects.

Governance

The UI SDK project is solely owned by the product development unit (PDU) "NAM", which also employs the core team and as such the UI SDK can be classified as governed by a single organizational unit. In some of the interviews, the ownership of the UI SDK was said to be the Ericsson Network Manager (ENM) project, which is a large product developed at PDU NAM, and as such ownership by ENM and PDU NAM can for the purpose of governance classification be seen as equivalent. Following is part of an answer to a question on who the project's stakeholders are, which highlights this single-organizational ownership:

> "Stakeholders will be both the ones paying our wages which is ENM at the moment, and our community." (U4)

The relation between the PDU NAM and the ENM is explained by one of the UI SDK architects:

> "My title is [architect] for PDU NAM. The role there is around supporting the development of PDU NAM products, primarily ENM." (U3)

Further solidifying the classification of the UI SDK project as owned by a single organizational unit is the following extract. This extract also mentions some perceived drawbacks of the single-organizational ownership, as well as how the UI SDK team wants to transition towards a less centralized ownership in the future, in order to mitigate these drawbacks:

> "In particular I think the perception of the UI SDK, [is] that it is something based out of PDU NAM. So, you know, there is that second barrier there that people don't want to contribute code to... outside of their own organization structure. So they don't necessarily feel ownership of the UI SDK. They feel like it is a bit centralized within PDU NAM. That is why we have to be more decentralized and more common if you like. And in that way we hope we get greater, you know, community wide, global, contributions, and rely less on ENM as a heavy contributor." (U3)

The notion that the UI SDK project wants to move away from the current single owner and towards a more decentralized and distributed project ownership model is also highlighted by the following snippet:

> "[...] we are in a bit of transition in terms of team make-up as well. We are starting to... Looking at other opportunities for more open model. But I guess one of the key things we are finding is that we still need a core team working on, sort of like the kernel of the SDK. And that is something that we haven't cracked yet. We would hope to work towards a distributed core team, decentralized core team, that is sort of our next evolution, and our next challenge if you like." (U3)

Objective

The objective of the UI SDK is to provide a high-quality framework for developing JavaScript-based user interfaces. The focus on high quality together with a fairly open contribution process makes the UI SDK best fit into the service-oriented class of inner source projects. Evidence for the aim of producing high quality software is seen in how the UI SDK core team regards backward compatibility and stability, exemplified in the following extract:

> "Because in the framework we guarantee backward compatibility, and since it is there... The only time we broke backward compatibility was in the case of a bug. So we consider something backward compatible if it's a full decision and we advertise that a new version will be available, or if something breaks and it was not intended to break like just some function just handles differently. In that case we handle it as a bug. But if it's an architecture change, we try our best to make it backward compatible. If we can't, then we would advertise that we will release a new major function, a bit like all the open source projects." (U4)

This commitment to high quality is seen as evidence that the UI SDK is unfit for the utility-oriented objective classification since such projects are unlikely to put much effort into providing compatibility with older versions. Between the two remaining classes, service-oriented and exploration-oriented, it can be seen that the UI SDK is, and has been, part of both during its development. For example, the next version of the framework is adopting new technologies, pointing towards an exploration-oriented objective:

> "So we are looking at a big evolution at the moment towards web components. [...] it's sort of a paradigm shift. If you started in a modern project with backbone.js and told people you were using backbone.js they would kind of scratch their head and say 'well, that's a bit 2012, 2013'." (U3)

This next version does not necessarily deprecate the old version though. In fact, the UI SDK team is committed to also support older versions, again strengthening the argument for an exploration-oriented or service-oriented classification. The following quote highlights the UI SDK's commitment to support older versions also once the next version ("version 2") is released:

> "We have this framework that we have been supporting for five years, and we expect to have a version 2 now. We have kind of watched the miracle and said: 'There is a big paradigm shift, let's go on the bigger trends and on the smaller trends'. And we are going to shift towards a new platform now. But we intend to support the old platform throughout the duration of version 2. So until version 3 in the distant unforeseeable future, we will have support for v1 and v2." (U3)

A differentiating factor between an exploration and a service-oriented objective is whether the project is open to external code contributions, or if the project is primarily developed by a smaller team. In the case of the UI SDK it is the former stance that is taken, as described by (U3) in the following quote, making the objective better classified as the service-oriented class:

> "There is a contribution process. And we dedicate a certain amount of capacity for supporting contributions. [...] There is a set of criteria, and again the more times you have been through it the easier it is to fulfill the criteria in terms of the tests you have to have, and quality of the code." (U3)

The classifications for both governance and objective were also confirmed by one of the project members in a response to a follow-up email, as presented below. This reply was in response to being presented with the classification framework definitions of Capraro & Riehle (2016) and our initial classification of the UI SDK project, and asked to provide their thoughts. However, it should be noted that the common differences in how strictly community contributions are controlled, as expressed in the underlying definitions by Nakakoji et al. (2002), were not explained:

> "I agree that the UI SDK has the 'Single organizational unit' governance status righty [sic] now but we are trying to move to multiple. Also, I agree 'Service-oriented' is probably the best fit objective even though any of these could be argued." (U3)

4.2 RQ1: How Are Features Prioritized In Inner Source Projects?

In the analysis of this research question, we found that feature prioritization happens in three steps: elicitation, delegating the implementation, and lastly the contribution from either the core team or the community.

1. **Elicitation:** In the first step, requirements and ideas for features are gathered from various sources. Such ideas can come from either the core team itself or the community. In the case of the core team, its members often have a long-term vision of in which direction they want to take the project. To what extent

this vision is validated with the community varies between the projects. In the case when ideas come from the community, requirements are gathered by the core team by talking directly to developers on site, seeking ideas from the forum, and through community meetings in which representatives of each major stakeholder group is invited to participate.

2. **Delegating the Implementation:** In the second step, it is decided whom of either the core team or the community should implement a given elicited feature. One of the most important factors is whether the feature will benefit multiple stakeholders or not. If it does, it may be important enough to be prioritized by the core team, and as such, the core team will implement it. On the other hand, if the feature is more niche and specific to one stakeholder, it may be pushed back to that stakeholder for her or him to implement. Another reason for the core team deciding to implement the feature may be that it is simply too complex for the stakeholder to handle, and it is better if the experienced (and full-time employed to work on the project) core team members perform the implementation. However, a feature may still be prioritized by the core team despite not meeting any of the criteria above if the feature has been escalated and is requested by upper management. Finally, it is noted that it is usually faster for the community member to implement the feature herself or himself, provided she or he is experienced enough.

3a. **Core Team Implements:** Depending on the outcome of the second step, different practices and techniques are employed. If the core team implements the feature, it is seen that the feature is simply put on the backlog and prioritized in an agile way depending on e.g. how many stakeholders seem to benefit from its implementation.

3b. **Community Implements:** The practices are more involved when the implementation is delegated to the community. First, as can be seen in both projects, the core team provides mentoring to the contributing community member to guide her or him in the right direction, and help with implementation and any questions that arise. Second, which is seen in JCAT, a technique called 1/3-review is used that urges contributors to submit an initial code skeleton for review before starting any real implementation. This has the effect of catching poor design decisions early in the process, and in the end reducing the time the core team has to spend on the final review. Third, both projects use code guardians, who are tasked to make sure that submitted code is of high quality. Finally, well-defined code guidelines have been set up to ensure both high quality and consistent acception and rejection criteria. Once the contribution has been submitted, the core team needs to make sure that the feature is of value, especially if it has not previously been validated with the core team. At this stage, the feature is checked to be in line with the core team's overall vision of the project and so that it aligns with the overall community's needs. If it does not fulfill the latter criterion, the core team may accept it anyway, or in some cases take it upon themselves to make the feature more general to satisfy a broader need.

4.2.1 Step 1: Elicitation

The first step of prioritizing features is the actual elicitation, in which features are collected from various sources. In both JCAT and the UI SDK, ideas for features and requests for features usually come from either of two sources: the core team or the community. While the most common source of eliciting features seems to be the community, the core team can also envision a roadmap of what needs to be done and, in JCAT, suggest this to the community at a stakeholder meeting:

> "You could say that the JCAT core team [...] who you could say suggest a vision. A bit of an overview, like 'we have to do this or this.' Then you put this forward at the stakeholder meeting and then people either vote or give their opinion or how they solved similar problems. But it's often a very general vision." (J4, translated from Swedish)

In the UI SDK, the overall vision decided by the core team is not explicitly validated in collaboration with the community in the same way as in JCAT:

> "So there is a core team. We do have a vision of what we want from the UI SDK, and what it should do, and what technology we should be moving on to. And still the core principles that we still have and want to maintain, about building simpler and very modular framework, that allows you to build your set of applications. So we have our core principles, that is why it is important to have a core team and have them understand what we are trying to do. And then you can evaluate people's requirements and see how best fit them with the UI SDK, and those principles and how it best serves the wider community." (U1)

As mentioned, while the core team comes up with an overall vision, the most requirements and requests for features come from the community. This seems to be done in a multitude of ways, with one of the most common being simply going out and talking to developers that use and/or contribute to the project:

> "There usually is someone... They go around to the different offices, or sites as they are called, and ask what the problem is for people. Then maybe someone says 'we have lots of trouble connecting to this server that is needed for this.' Then maybe we say that we will add support for running a local server." (J1, translated from Swedish)

> "[...] physically visit some of our bigger sites, some of the bigger products, that are using the UI SDK. Talk through with them what their requirements are and see how can we take their requirements and build them into something that can be useful for all." (U1)

The UI SDK core team also arranges regular community meetings and workshops, both to show and get feedback on new features, and to put community members to work in researching new ideas:

> "So this is where a community evangelist and mature users and early adopters are key as well, so now we try to involve them in more workshops and get their opinion, and get them to start researching and looking into the best possible solutions for things, etc." (U1)

Similarly in JCAT, a monthly stakeholder meeting is held in order to collect thoughts on current issues and how to move forward:

> "Then each month we hold a JCAT stakeholder meeting where we talk about what is hot for everyone and how the inner circle of JCAT development is planning on moving forward, to let stakeholders have opinions on what is important to them right now." (J4, translated from Swedish)

Several of the interviewees, both in JCAT and the UI SDK, bring up project-specific discussion forums as a way of getting ideas for features. There is an emphasis on achieving an open and welcoming atmosphere on the forums, where anyone can have their say:

> "We are very careful to not be aggressive or critical on the forum, rather we take in all the ideas we get, and it's open and nice environment. That people feel that they dare to write. It works well." (J1, translated from Swedish)

> "In a normal scenario we communicate with the community via the forums that we have. There is a section in each one of the forums for the libraries for recommending or introducing new requirements for the libraries." (U2)

4.2.2 Step 2: Delegation of Implementation

Once requirements and corresponding features have been elicited from either the core team or the community, it needs to be decided who is going to implement the feature. This is a key point in how inner source differs from traditional corporate development; if a task cannot, or even should not, be taken on by the core team (see 4.2.3 Step 3a: Core Team Implements), it can be offloaded to the community (see 4.2.4 Step 3b: Community Implements). Selecting who should implement a feature depends on many factors such how niche the feature is, whether the requesting community member is experienced in implementing such a feature, among others.

One of the most deciding factors for if the core team should implement something is if it will benefit many stakeholders:

> "So basically if a lot of members in the community require a feature, if we think that one feature would benefit a lot of members, we would prioritize this feature." (U4)

Another valid reason for the core team to do the implementation is if the feature is complex enough that a contributing user has neither the time nor the experience to do it, the core team will put it on their backlog instead:

> "If it's a complicated thing that requires a lot of architectural work, then people will generally realize that 'eh this isn't worth trying to do for us, we'll write it on the architects instead, or on some of the dedicated developers.' " (J1, translated from Swedish)

> "Or if it is something really complex and the person could have the skill, but we see that 'okay it's very complex,' if someone not in the team does this, we will know how it works. Like we could as well take it ourselves." (U4)

However, since the core team often already has a full backlog when new features are requested, such features will be put on the backlog and dealt with at a later time. In the case where the situation does not fall under any of the criteria listed above for when the core team implements a feature, and it is of importance that the feature is implemented on short notice, the request must come from upper management. Such a situation is deemed by the JCAT team as a failure from some involved party, and should ideally never arise:

> "If we are to take care of something now, it will have to be escalated as it's called, that is it needs to come from a high boss saying 'we need this right now.' Things like that should never happen and we view every time it happens as a failure, maybe not always from our side, but from somewhere." (J1, translated from Swedish)

It is also noted, in relation to the previous quote, that it is generally faster for a community member to implement something him or herself rather than requesting the feature, provided that the implementing developer has the required expertise:

> "So generally it's always faster for people to contribute themselves if they aren't totally out in deep waters." (J1, translated from Swedish)

4.2.3 Step 3a: Core Team Implements

In the case where it is decided that the core team should implement a feature, not much changes compared to if the feature came from the core team itself. Generally, the feature will go on the backlog and will then be prioritized in a seemingly agile fashion according to, for example, how many stakeholders need it:

> "[...] if you have some community members requesting for a feature and we decide to take it, it would go to the backlog. Then either if it's on the bottom of the backlog and we lost sight of it, it means things getting piled up on the top... yeah, that happens. Or it's something quite major

> or that could benefit a lot of people, we try to keep it to the top. So within a couple of sprints time it would be released." (U4)

> "We have to remain very agile. While we do have a roadmap of where we would like the UI SDK to go, we have to be very mindful that we have to respond if someone has a very urgent need for something. And we have to be able to plan that into the next sprint, or take it in the current sprint." (U1)

In order to show off the features that are implemented, both projects organize demo meetings which stakeholders and users can attend:

> "[...] these community practice meetings that I talked about. For example we have a number of large user groups that have meetings every week where representatives for all user groups call in. We also sit there and we can present some new stuff." (J1, translated from Swedish)

> "We run a demo every three weeks, and we work in a three week sprint cycle. Our demo would be well attended by our stakeholders. So they would come along for 15, 20, half an hour at most, and see what is new and what we have been working on, and what contributions we have gotten." (U1)

4.2.4 Step 3b: Community Implements

There are primarily two cases in which the community will contribute a feature: if it was pushed back to the community member by the core team (through step 2), or if it was dreamt up by such a community member in the first place, without first consulting the core team. No matter which is the case, the same practices are employed by the core team and the same principles apply for acceptance of the resulting contribution.

Practices

Both projects employ some type of mentoring to contributors, partly as it has been seen that a difficult part of managing the community is to compel developers to make their first contribution; once the first one has been made, it gets easier:

> "It's not unusual that people who have made a contribution come back to do another one. But making people do the first one, there is the problem." (J1, translated from Swedish)

To get around this, both teams seek to make the threshold for contributing as low as possible, for example by mentoring:

> "There is also documentation for which the architect has the best knowledge of certain areas, so if someone comes in and is going to contribute functionality related to [my area], they talk to me. And if they have something to do with [another area], they talk to [my colleague]." (J1, translated from Swedish)

> "[We] dedicate a certain amount of capacity for supporting contributions. So it's not something that, you know, that usually can be run. Unless a team, outside the core team, is very mature and has previously delivered several contributions, they are going to need a lot of support walking through that process." (U3)

> "Then, if the person doesn't have the skill or if it's something which is touching the barebone which is something that... If it could possibly go wrong, we would try to mentor this person as close as possible so that we really don't meet any unforeseen side effect." (U4)

On top of mentoring contributors, the JCAT team employs a technique which they label 1/3-review. When a contributor has come up with a rough sketch of both desired functionality and a program skeleton, they should contact the JCAT team to verify that they are on the right track:

> "What we are trying really hard now to solve the problem [of users contributing poor features] is with something called 1/3-review. [...] essentially it's when you have finished the analysis and perhaps coded a small code skeleton, that we need these classes etc. Before you have built everything. Then we try to get involved so that we steer it, so that when they have eventually implemented everything, it's a much faster process to go through it here. There are very few that come to review and we say 'no you can't put this in', because we have solved it much earlier in most cases. So that essentially never happens." (J1, translated from Swedish)

Both projects have trusted lieutenants (called code guardians) who help in reviewing and maintaining overall code quality. It seems that in JCAT, these guardians (or specialists as J1 calls them below) are each responsible for a part of the code base, while in the UI SDK the guardians have a collective responsibility:

> "It turns out pretty quickly that with a million lines of code, you can't take on the whole tool. Instead we have specialists. My speciality is for example [this area], not so much the product side because I am quite uninterested in that." (J1, translated from Swedish)

> "Because as the core guardian of the UI SDK, one of my... part of my work is ensuring is that every component that is contributed to the UI SDK maintains a quality standard, and also that it's useful for everyone and not only works in a particular use case." (U2)

Furthermore, both teams have rigorous code guidelines, enabling consistent feedback across contributions:

> "[...] we have rigorous guidelines for how code should look. If you are throwing an exception in Java you should throw it like this, this is how you log it and such. The fact that we have this written down enables us to be very consistent towards people who try to contribute code. They kind of do not get different comments each time they contribute; it's the same comment. If you break the same rules you get the same answer." (J1, translated from Swedish)

> "There is a set of criteria, and again the more times you have been through it the easier it is to fulfill the criteria in terms of the tests you have to have, and quality of the code." (U3)

Acceptance criteria

Once a contribution has been submitted, the core team will make a decision whether it should be accepted, rejected, or if it is in need of modification first. First, features that already exist will be rejected:

> "What can happen is that someone skips [1/3-review] and then comes with new functionality, and we say 'this already exists.' That has happened a few times. It's always sad when it happens. We cannot do much more than make it clear to everyone that 1/3-review is the standard [...]" (J1, translated from Swedish)

Second, contributions that are not in line with the overall product vision will also be rejected:

> "[...] it needs to be in line with our direction. So for example, if someone wants to add support for a tool which the board agreed that we are not to that direction, then of course it won't go through even if the code is good quality." (J5)

> "The only reason we might not want a feature is that it doesn't match our UX, it is not compliant with our quality standards, [or] it is non-compliant with backward compatibility like I explained previously." (U4)

Third, contributions should ideally be in line with what the project community in general needs.

> "[...] when [someone] has developed a component that will be useful for the community to have, what we would do is to encourage this person to actually look at their code and make sure that it's done in a generic way." (U2)

As a part of aligning contributions with what the community wants, the core team can at times help with this, likely because it may naturally be difficult for an individual community member to know what type of functionality the community as a whole is the most in need of:

> "So we are trying to encourage people to do more themselves and contribute it back into the UI SDK. And in some cases, they will do it, but it will be very much for their product. The work we have then is making it more widely appealing, or you know taking out the more product specific requirements that they put in or something and making sure that it works for everyone." (U2)

> "When people actually [contribute] I would say the most common thing is for the responsible for the project is to look at the review and often not merge it straight away, but rather [clone it], patch it a little, re-write it so it fits in the context, and then maybe merge." (J4, translated from Swedish)

4.2.5 Discussion

An interesting difference between the two projects is that the UI SDK does not use a stakeholder board for deciding a long-term vision, while JCAT does. We speculate that this difference could have an impact on how applicable the vision that is decided upon is on each individual stakeholder; if the core team does not decide the project's long-term vision in consultation with the community, as is the case to some extent in UI SDK, it is possible that requirements from the community are lost to a greater degree than if the opposite was true. On the other hand, we believe that having a stakeholder board where each stakeholder has his or her say might lead to a larger spread of features and less certainty about what to do next. In the same vein, not having a stakeholder board and instead mainly relying on a single or a few stakeholders may result in a clearer vision which could make for a more cohesive product.

A possible explanation for why JCAT has a stakeholder board while the UI SDK does not might be because of differing governance structures. The fact that JCAT is governed by all organizational units while the UI SDK is governed by a single organizational unit is supported by our project classification (see 4.1 Project Classification), and may explain why JCAT needs to take more opinions and needs into account when deciding on a way forward while the UI SDK primarily needs to adhere to the requirements of its main stakeholder ENM. Of course, this does not mean that the UI SDK ignores the wishes of its community, but when deciding on what to do next, ENM may be more important to take into consideration.

The practice of 1/3-reviews appears to work very well for the JCAT team, and seems like a practice that should be widely adopted. As already stated, it has the effect of

catching contribution of unwanted features and those who are poorly architectured before too much work is done that might be thrown away in the end. Moreover, employing 1/3-reviews has a number of additional positive effects. First, the JCAT team rarely has to reject features at the final review, which reduces the total review work that the core team has to do as well as the amount of refactoring that a contributor has to do due to poor architecture that needs to be revised. Second, contributors waste less time on features that will never be accepted due to not being in line with vision, already existing in some form, being too niche, or some other reason.

In terms of communication channels, it was found that both projects make thorough use of forums as a way of giving support and gathering requirements from the community, among other activities. This seems to imply that the findings by Fitzgerald (2011), that forums are critical for open source projects, are transferable to inner source projects.

Furthermore, the results for this research question confirm the observations by Gurbani et al. (2010) that a common structure of inner source projects is to utilize trusted lieutenants (called code guardians in the explored cases) to monitor and ensure quality of software components.

4.3 RQ2: How Are License Issues Related To The Usage Of Third Party Products Handled In Inner Source Projects?

The findings for the second research questions is presented in this section. The section is structured into two primary themes, briefly summarized below and further expanded upon in their respective subsections, followed by a discussion on the results.

- **Inner Source Project Implications:** it can be seen that there are certain implications for inner source projects when it comes to third party products and the legal considerations required by the associated licenses. Specifically, it can be discerned that the case projects do not seem to be required to handle licenses as strictly as a product project would be and that it is the users of the inner source project that are ultimately responsible for the licenses.

- **Using Third Party Products:** it can be observed that the two projects take steps to make sure that third party product dependencies are handled carefully if included.

4.3.1 Inner Source Project Implications

It is seen in the analysis that being an inner source project implies two specific differences in how legal aspects of licenses are considered, compared to product projects:

- **License less important:** it is observed that it is of less importance to handle licenses of third party products in inner source projects compared to in product projects.
- **Users responsible:** it is seen that inner source projects cannot take responsibility for their usage of third party products when the inner source project is used in a product. Instead, the product using the inner source project must assume that responsibility.

License less important

In general, licenses associated with usage of third party products are less of a concern for an inner source project, precisely because the project is for internal use only. However, there are still benefits in making sure that projects handle third party products carefully, as doing so can prepare the project for potential delivery to external customers in the future. The general concept is described by a JCAT upper manager:

> "What we're doing here is inner source, which means we have different requirements for trade compliance than the products [Ericsson in general] are selling." (J5)

The less strict need for handling licenses of third party products is further expanded upon by the same JCAT core team member in the quote below. The quote also mentions the fact that the core team errs on the side of caution. Furthermore, the cautious approach to third party products taken by the JCAT team is thought to make releasing the software outside of Ericsson a possibility also from a legal standpoint.

> "Our situation as it is today is that we do not release binaries outsides of Ericsson, still, we need to fulfill all legal requirements when it comes to 3PP handling. In practice, it means that we're very careful with onboarding 3PPs and totally avoid restrictive licenses such as GPL." (J5)

The same cautious approach is also found in the motivation behind voluntary trade compliance audits performed by the JCAT team, as explained by a top manager:

> "As I said, since we are an Ericsson internal tool, it doesn't have like strong central trade compliance audit, we do it for ourselves just to make

sure that everything is fine.“ (J5)

A concrete example of why licenses are handled with greater care than what is perhaps strictly required is provided by another JCAT team member. The example again highlights the fact that careful license handling can benefit the project long-term, should it ever be used externally:

> “I have even seen commercial licenses, which wasn't a problem as we had paid for the license. But we wanted to sell the code to a customer, which made it impossible to sell the code as there was this commercial dependency that we had the right to use but not sell.” (J4, translated from Swedish)

Users responsible

Another implication observed in the analysis was that inner source projects cannot absorb the responsibility for the third party products that are used within their project; if an inner source project is used in a product, it falls on the product team to ultimately verify that the third party dependencies of the inner source project can be used within their product. Furthermore, it seems to be actively discussed whether it could be changed so that the inner source project could, in fact, take on the responsibility for their third party product usage.

In the following quote, a member of the UI SDK team likens the relationship between the UI SDK and the products using it to the one between Lego bricks and a finished Lego model. The UI SDK is used to build products, but it is itself never sold outside of the company. On the other hand, the products using the UI SDK are in fact sold externally and as such must carefully disclose and handle licenses for the parts that were used to build the product. The UI SDK team can and do still help in the process of satisfying legal policies by making it clear what third party products are used in the project, but the ultimate responsibility remains on the product team using the UI SDK.

> “So basically we are the bricks and the applications are the model that you build. When you're going to sell the house or an airplane or your car, you're saying ‘what is inside?’ So as we provide the bricks, we just say the bricks are using these libraries, but we are not selling the product, the framework so far is only internal. However, all the applications using the bricks should put in their documentation and should basically satisfy the legal requirements. So it means, document the licenses properly etc. We just make all the licenses, as you can see in our documentation, totally transparent.” (U4)

Another UI SDK team member expresses the same sentiment in the quote below.

The extract mentions the FOSS evaluation process[1], which is an Ericsson process related to verification of third party product license handling. FOSS in this context may also be used in its more common way: an acronym for free open source software. It will henceforth be clarified in which instances the FOSS evaluation process is intended. Also mentioned in the quote is the benefit of this process becoming easier as the inner source project is better recognized:

> "Not only that, but if you want to use the UI SDK, including its FOSSes, you have to do your own FOSS[-evaluation] is my understanding. Because there are other users of UI SDK, you can point to precedence there and say 'look, it has been used here. It is low risk.' " (U3)

The same project member again confirms that the UI SDK is not currently responsible, and discusses their ambition of trying to absorb this responsibility in the future:

> "Yea we have some discussions about this with some of the [people related to the FOSS process]. Because at the moment the UI SDK is not responsible. We wanted to take that responsibility away, it's one of the things you want to do, or the main thing you want to do as a framework developer, is to make things, make life, easier for your users right. So we wanted to centralize license handling so that every user of our framework didn't have to do their own FOSS-analysis. But we didn't achieve that so far. I believe the current state of play is that whoever wants to use something has to do a [FOSS evaluation of] their particular project." (U3)

4.3.2 Using Third Party Products

It can be seen that both of projects have processes in place for proper handling of third party product licenses. In essence, there are two different ways in which the projects handles the usage of third party products:

- **Avoid third party products:** By attempting to avoid third party products, and instead creating the functionality internally in the projects, it is possible to circumvent much of the license handling otherwise required.

- **License-aware usage of third party products:** Although not including any third party products might be the ideal solution from a licensing standpoint, it is sometimes hard to avoid all such third party dependencies. This calls for practices and methods for how licenses of such dependencies should be handled when the dependencies are included. In our analysis, we see that the responsibility for this third party license handling is primarily put on the core team.

[1] Ericsson at all times strictly follows the legal requirements for FOSS license handling.

Avoid third party products

One way to avoid having to deal with license issues due to third party products is to simply not include such software in the first place. This is of course not always a simple task; third party products are usually included for a reason. Nevertheless, removing third party dependencies is what the UI SDK project has been attempting. The opposite is observed in the JCAT project, where third party dependencies are often preferred. In the case of the UI SDK, replacing dependencies on third party products with code of their own also has other benefits according to the interviewees, such as reduced size:

> "Well, we kind of have a philosophy of build rather than buy. Which is kind of an Ericsson-wide, I think, technology philosophy. But we try to not include [3PPs, Third Party Products]. When we started, we would have jQuery so we could use jQuery libraries. But overtime we actually have swapped that out and we no longer depend on jQuery. Because it is a really big, heavy, library. And you are bundling that with the UI SDK, and we just felt like we weren't using enough, and there were things that we could do ourselves." (U1)

Two other members of the UI SDK team echo the same idea of, if possible, building solutions themselves rather than using third party products:

> "But as we are fairly established at this point, it's not very frequent that we say 'let's use this 3PP' if you get me. If it is something that we are actually able to implement ourselves, we do it." (U2)

> "**Interviewer: So you kind of tried to avoid using libraries if you can?**
>
> U4: Yes exactly. If something is available as native, why would you use a wrapper?" (U4)

The JCAT project does the opposite, preferring third party dependencies over implementing it themselves:

> "Exactly the opposite for JCAT. First check what is available as 3PP, then implement remaining." (J5)

License-aware usage of third party products

Although there seems to be a concerted effort to avoid using third party products in the UI SDK project, both the UI SDK and JCAT still occasionally decide on relying on such external dependencies. The projects primarily put the burden of properly handling these dependencies on the core team. In the UI SDK team, it is the trusted lieutenants (or code guardians as they are called in the UI SDK) who take on the

responsibility of ensuring that licenses of included third party components are usable within the component that they oversee. In JCAT the responsibility is instead put on the upper management, who in turn performs audits to ensure license compatibility. Both processes rely primarily on an Ericsson-provided global tool called the Ericsson Software Bazaar (Bazaar for short), which is an internal database of third party software, the licenses they have associated, and a recommendation for whether each license is usable in Ericsson software. Finally, build systems that allow for explicit listing of dependencies, such as Apache Maven, are mentioned as helpful in the license handling process.

That the responsibility for third party product handling in the UI SDK project is put on the code guardians, is explained by a top manager in UI SDK:

> "I suppose we have code guardians in the UI SDK. And we would review everything, so we know everything that is in there." (U1)

Another member of the UI SDK core team expands on the same topic, including the entire core team in those responsible and clarifying that it is during the code review process that licenses are verified:

> "Code guardians, architect. Back in the team, we are responsible. Or we judge ourselves responsible. Basically we will not accept code contributions where the license is not clear." (U4)

The JCAT team performs audits of the project. One trigger for these audits is that some user from the community expresses a desire to release the framework to a customer:

> "Actually it usually happen as soon as anyone mentions 'I would like to release [MJE, a derived version of JCAT] to a customer, is that okay?'. Then we usually check the licenses [...]" (J1, translated from Swedish)

Audits are also performed in the JCAT project before new third party dependencies are added:

> "For [JCAT] (where I'm the responsible person), we always perform audit for all new dependencies." (J5)

The responsibility is also handled differently in the JCAT project. Instead of the shared responsibility assumed in the UI SDK project, only the top manager is seen as responsible in the JCAT project. This is observed in the following quote by J5:

> "The second thing is that for the JCAT framework, I mean the core part and the community-managed common libraries, those which are mainly maintained and developed by the JCAT program, for that part I am the responsible as [upper manager]." (J5)

Both projects heavily rely on the Ericsson internal Bazaar tool for support in deci-

sions regarding licenses. The tool provides usage recommendations for third party products in regards to what licenses they have:

> "There is a project called Ericsson Software Bazaar which keeps track of a lot of third party libraries. They keep track of if [the third party libraries] have reasonable licenses and gives recommendations if it should be used or not. Keeps track of different versions and such." (J1, translated from Swedish)

Also build systems that explicitly lists dependencies are seen as important, as they simplify the process of finding used third party products. This is explained by a JCAT member as follows, where Apache Maven is one such build tool:

> "What [third parties products] we use is easier [to see] as we are using Maven. Then it is possible to see all dependencies there." (J2, translated from Swedish)

4.3.3 Discussion

The first part of the results for this research question describes the implications associated with being an inner source project, in regards to license issues with third party products. We believe these implications to be heavily associated to the case company and that these implications, therefore, are unlikely to be shared among inner source project at other companies. This belief comes from that the implications seemingly are a result of how the Ericsson FOSS evaluation process is implemented on a more global level. Unfortunately, the hypothesis that the implications are due to Ericsson processes could not be verified within the scope of the study.

When it comes to handling licenses of third party products, it seems like the two projects do this in somewhat different manners. In the JCAT project audits are performed primarily by the upper management, whereas the UI SDK project makes the license check a part of the code review process, performed by its trusted lieutenants. Having the trusted lieutenants perform the reviews means the UI SDK might potentially need fewer roles involved in accepting contributions. However, doing so puts an additional burden on the trusted lieutenants in that they must also be able to evaluate licenses themselves. Having such legal knowledge is perhaps not as common for developers used to the more corporate style of development at Ericsson, where license handling is generally left to a specialized license handling team.

Another difference between the two projects is their stance on usage of third party products. The UI SDK project attempts to limit their reliance on third party dependencies as much as possible, whereas the JCAT project prefers using dependencies over building the functionality themselves. A possible explanation that we see is the fact that the JCAT project is built on third party projects already from the start (i.e. JUnit and TestNG), and as such fully excluding external dependencies might

never be possible. In such a case, where including some third party products is close to unavoidable, we speculate that including also some additional dependencies is less problematic as the need for proper handling is already mandated by the critical third party dependencies. In the UI SDK, it might be more realistic to fully exclude third party dependencies, allowing the project to avoid many of the license issues that are associated with usage of third party products.

4.4 RQ3: How Can An Inner Source Project Spread Awareness Of Its Existence Within The Company Where It Is Developed? (Discoverability)

This section aims to answer the third research question. The main findings are listed below and are expanded upon in their respective sections:

- **Project finds people:** the core team actively promotes and evangelizes the project to expand the community.
- **People find project:** developers organically find the inner source project by hearing about it from others.
- **Top-level support:** being supported and recommended by management helps to spread awareness of the project's existence.
- **Inner source program:** an inner source program providing common infrastructure and time allotted to working on inner source projects is desired.

4.4.1 Project Finds People

It can be seen that evangelism by the core team plays an important role in establishing and expanding the community surrounding an inner source project. This active search for people seems especially important while the community is somewhat smaller, and seems to lessen as the project community grows and is able to help spread the project. However, both the UI SDK and the JCAT project still engage in various activities to attract new users and contributors, in spite of both having large communities already.

An example of an activity performed early on by the UI SDK team was on-site demonstrations:

> "Actually that is a huge body of work that we did when we started out. And we used to travel. Members of the core team used to travel to other

> product development units in Ericsson, so we have been all around the world. To go and give a hands on demonstration of the UI SDK. Show them how easy it is to use, and how much faster their development would be, etc., etc. So you basically market the advantages of this product to product managers and developers." (U1)

> "Around the time, sort of, the first wave of UI SDK, Version 1. when it started to become something useful. We did a bit of a of a roadshow. It was actually not so much myself but I did a bit as architect. I was more involved in the very early evangelism around, you know, why we needed the framework and so forth. And then I get involved whenever there is discussion about common components and things like that." (U3)

Also in the JCAT project the management actively seeks to promote the project to anyone who may not be using it already:

> "Then also we are out doing promotion, that's primarily [J5]'s job. As soon as he finds out that there is someone out there doing test automation and not using JCAT, he's there talking to them." (J1, translated from Swedish)

In both JCAT and UI SDK, onboarding is also performed as part of the on-site promotion with organizations that are interested in adopting the software:

> "Then we run a short pilot. We have this professional services team who can go, these people can go on site and do a proof of concept for that product. Creating the first test cases together and see that it fulfills the products requirements in terms of test automation." (J5)

> "We can be asked for support around the organization, so for onboarding workshops we create two videos to release a lot of this work so that we record one video and view the video instead of being there every time." (U4)

Another way of actively spreading awareness is through hackathons (programming-related events usually centered around a predetermined theme), which in the UI SDK is run by the core team:

> "Within the actual building here we conduct hackathons and workshops [...]" (U2)

4.4.2 People Find Project

As the project grows it becomes more common for people to find the project in contexts other than the project actively marketing itself to them. The mechanisms for this spreading are essentially the same for both JCAT and UI SDK, and mostly

rely on that some employee that has experience with the projects mentions them to other colleagues. For example, people may be in a meeting with a UI-designer who says that the component being shown on the screen is part of UI SDK, thereby planting a seed in the meeting participants' minds that UI SDK may be a useful tool:

> "We are working very closely with UI-Designers. So when a UI-designer shows a UX-screen, they will say this is available using this component in the framework." (U4)

People involved in the projects and their communities are further encouraged to spread the word of the project to other members of their organization:

> "[...] we of course always encourage the board members to share their experience with JCAT among the other products in their areas. So that is how usually people get to know about JCAT and this is how the word spread." (J5)

Furthermore, the people that like the software will naturally spread it on to their peers, asking why they are not using this great product:

> "[my colleague] is for example such a person that would absolutely consider that if she finds someone who is doing test automation without JCAT, she would ask the question 'why aren't you using JCAT?' Since it's good enough for people to like it, it spreads itself actually. But you also need people like [J5] and [my colleague] and such that drive it." (J1, translated from Swedish)

Since JCAT is the largest inner source project in Ericsson and has been around for a long time, it also naturally draws attention to itself. The somewhat unconventional nature of the inner source development process also plays a role in attracting interest:

> "[...] they naturally draw themselves to JCAT precisely because it's the most growing test automation framework at Ericsson. It's hyped right now, so it comes by itself. Developers naturally become interested in it because we have a bit of a different model of how we work, instead of this traditional 'we have a budget, we follow the budget and made a product'." (J4, translated from Swedish)

Another point that is brought up in regards to having people find and use the projects is the importance of keeping the software at a high quality level at all times and focusing on what is important for its users. If users find the project and find out that it is of subpar quality or does not solve the problem that the user may be having, it will not be put to use.

> "I would say, a project doesn't advertise itself [...] When an application is going to be developed you need to spread the word that using this framework you will make. [...] The framework allows you to have dif-

> ferent very key principles. First we guarantee that everything we have on the framework is brand compliant. So all the UX that we use, all the buttons, colors, etc. is brand compliant. So you will never have to question; if you use a color, is this the right color, if you use a button, is this the right shape. So we provide this." (U4)

4.4.3 Top-level Support

One aspect that could be very impactful for discoverability is top-level support, which both projects say they have received. Specifically, both projects have been selected as the recommended tool for their respective area by executives high up in the organization. This made usage of the projects more common, and attracted new users and contributors:

> "[...] we have actually had a decision from the highest boss at Network Products, which has 12,000 employees under him, that if you are doing test automation then [MJE, a customized version of JCAT for the Network Products division] is the one to go with. And MJE is based on JCAT. That makes it so that the poor people that want to keep using some other tool has to answer to the same silly question over and over again: 'Why aren't you using MJE?' [...]. It has the effect that when we say 'Can't you use MJE?' and they say 'No,' then we say 'but the highest boss said that we should do it, so now we'll have to talk about this'." (J1, translated from Swedish)

> "It's something that we have been benefited by the actual mandate through management that actually recommends using the [UI SDK]." (U2)

In both cases, the support was not something that was actively sought by the projects, but rather something that was given to the projects for their perceived quality and the benefit of them being inner source:

> "They have realized it on a company level, they did when they merged these new network products, earlier there were a lot of small organizations... It's terribly expensive to have a lot of employees doing approximately the same thing. So we, who were the biggest, said that we can become even bigger and swallow all these people and become very big instead. So they said 'okay, go for it!' " (J1, translated from Swedish)

Monetary compensation based on the number of users was not observed in any of the projects:

> "[JCAT] had more-or-less same budget when it had 100 users as it gets now with 3500." (J5)

> "**Interviewer: Do you get extra budget the more users you have, and like that?**
>
> U3: No. Not at this stage." (U3)

4.4.4 Inner Source Program

Another aspect that was brought up multiple times was the need for a centralized inner source initiative at Ericsson. Currently, JCAT and the UI SDK are two separate projects with two separate communities, tools, etc. surrounding them. The interviewees generally see the lack of such a centralized repository as something that negatively impacts discoverability of various inner source projects at Ericsson:

> "I think what we are missing in Ericsson is an inner source portal. Or a platform. It is something we have spoken about before here, but we just, again, we don't have the bandwidth to kind of drive this or to build this." (U1)

The Google concept of 20% time is also brought up in multiple interviews. The purpose of 20% time is to allow employees to spend up to 20% of their time to work on tasks that are not directly related to their primary task. This 20% time is seen as something that could help in giving the inner source projects additional contributors, as the current perception is that most employees are too busy with their ordinary work tasks that they are unable to contribute to other projects.

> "I think it is an area that is lacking. I had thought in the past about things like, you know, Google's 20% time. If we had something like that, where developers were encouraged to spend 20% of their time, and you know in their [yearly performance goals]. Be given scores for having fulfilled their 20% time as inner source contributions. [...] to make something more generally usable it could take 3-4-5 times as much effort to make something generally usable than to have something that you can hack into your own particular application. So the extra overhead requires a lot of effort, and teams often don't have time for it. And they don't have any strong motivation to share something." (U3)

4.4.5 Discussion

It is interesting to see that both projects seem to have gone through similar phases of how awareness of their respective projects are spread. At the outsets of the projects, they had to rely primarily on their core teams for marketing, for example by doing on-site demonstrations. However, as time has passed and the projects have grown in size, they seem to put more and more of the marketing responsibility on the community. This result is not unexpected; the larger the community of an inner

source project grows, the more people will potentially talk to their peers about the project they are involved in, in turn potentially recruiting new developers, users or even organizations to the inner source project's community.

It may be tempting to draw the conclusion that a primary cause of the UI SDK budget not being correlated to the amount of users or contributor it has is that it is classified to be governed by a single organization, as covered previously. However, as can be seen in the findings for this research question, there exists no such correlation for JCAT either, which is governed by all organizational units. Therefore, no such conclusion in relation to the governance classification can be drawn.

As mentioned in the related work section, previous research by Riehle et al. (2009) suggests that one of the benefits of implementing inner source is that an organization can enjoy broader support and buy-in of projects. This is confirmed by the findings of this research question, as both projects have received support from upper management and have in fact been chosen as the recommended tools for their respective domains in the divisions where they are used. A possible explanation for why inner source development practices could lead to the project receiving support from upper management might be that when the community grows, the collective adoption of the software and experience using it also grows. Once enough developers become accustomed to using a tool or other piece of software for some purpose, it logically does not make sense for top management to advise or recommend another tool, as such a recommendation would require all developers using the inner source tool to learn a new tool, costing valuable resources. Furthermore, since one of the core aspects of inner source is that essentially anyone can contribute to the project (Stol et al., 2014), the software will evolve according to the developers' needs and possibly become ubiquitous. This notion has been observed by Capraro & Riehle (2016), suggesting that inner source enables "bottom-up collective intelligence," which according to Capraro & Riehle (2016) can translate into community members making the inner source software more fit for their needs by making contributions.

The need for a central repository platform is something that was brought up by many of the interviewees and is also part of in the infrastructure-based model of adopting inner source as described by Gurbani et al. (2006). In general, it seems that the infrastructure surrounding inner source development at Ericsson is somewhat lacking, with no central portal to host inner source projects (which can also provide marketing opportunities) and no centralized solution for communication in such projects like forums. It should be noted that while JCAT and UI SDK both make good use of forums, they have both researched and implemented the use of their respective forum software (they use different ones) inside the projects rather than having been recommended by management to use an Ericsson-supported solution. Another point related to the infrastructure of inner source development is that there is no time specifically allocated to work on inner source projects from a global direction, for which the 20% time as implemented by Google Whittaker et al. (2012) is brought up as a possible solution by many interviewees. Since one of the most hindering factors for involvement in inner source project seems to be a lack of time, such a solution could indeed be worth considering.

4.5 Validation Workshop

In this section, the outcome of the validation workshop is presented. The three areas in particular were discussed, presented below:

- **Inner Source Developer Responsibilities:** The members of the workshop see additional responsibility put on the developers of an inner source project.
- **Inner Source FOSS Evaluation Process:** The aforementioned FOSS process (an Ericsson internal process for evaluating license compatibility) is also brought up in the workshop. It was discovered that the inner source projects might be able to absorb more of the burden of this process than what was thought from the case study.
- **Inner Source Platform:** The current progress of developing an inner source program is presented. The implementation likens that of an infrastructure-based inner source program (Gurbani et al., 2010), but is limited to one division within Ericsson.

4.5.1 Inner Source Developer Responsibilities

During the workshop it was mentioned multiple times that developers who work with inner source software will be required to take on additional responsibility. In particular, discussion centered around the need for developers in inner source projects to be aware of licenses and the associated legal issues. Workshop participants indicated that the reason for this need is the possibility that the inner source software might be exported to a customer in the future. The AAT project (the project that served as discussion point for the workshop), in which selling to an external customer is already part of the roadmap, was presented as a potential inner source project, and is as such a project where developers would be required to consider these additional aspects from the start.

Discussion

The discussion during the workshop mirrored much of what has already been mentioned in 4.3.3 RQ2 discussion; when exporting an inner source project to an external customer, legal requirements become more strict. During the workshop it was generally thought that the developers of the inner source project should be the ones responsible for these requirements. Precisely who the developers are and what methods would be used to control the legal issues was never further specified. Although putting the responsibility on each individual contributor might be possible, we believe that this would be both impractical to implement in practice and that it would increase the barrier of entry for new contributors. Instead, it seems to us like this

responsibility should be absorbed by the core team, as is done in both the JCAT project (through audits) and the UI SDK project (during code reviews). However, a difference between the two projects investigated in the case study and the AAT project discussed during the workshop is that the AAT project already involves an external customer.

4.5.2 Inner Source FOSS Evaluation Process

The Ericsson FOSS evaluation process, i.e. the process used internally at Ericsson in order to validate license compatibility of third party products before they are exported to external customers, and how it relates to inner source projects were two of the primary concerns expressed by the members of the DevOps organization. Currently, it is observed that inner source projects are not doing these evaluations; instead, product projects have to apply for the FOSS audits also for the inner source projects they depend on. As an example it was mentioned that evaluations have been performed for the UI SDK by request of the AAT project, which is currently including the UI SDK project as a dependency. It was further explained that reviews of dependent projects would speed up if inner source projects applied for FOSS evaluations of their own. This increase in speed would come from that the evaluation process of the dependent project can then rely on the results of the already performed evaluation for the inner source project. It is also emphasized that the FOSS evaluation is a time-consuming process, especially if done for the first time, and that possible time savings would be beneficial.

Discussion

It can be seen that what is said by the workshop participants contradicts some statements made in this thesis's interviews in that FOSS evaluations are allegedly both possible and advantageous to perform by inner source projects. It should be noted that the people interviewed for the case study never asserted that this was the case, rather they assumed that this was how it worked. We see this as an additional indication that introducing a global inner source program could be beneficial, as we believe it would make it easier to spread best practices, such as performing FOSS evaluations, to the entire inner source community.

4.5.3 Inner Source Platform

A presentation was held on how inner source infrastructure is currently provided within the division on a more technical level. It was seen that there is no global effort to provide an inner source process and that the technical infrastructure used for other projects was not suitable for the collaborative nature of inner source. The mentioned technical limitations primarily concerned network firewalling and

partitioning, making it impossible for employees outside an organization to access projects within it. In order to solve these problems, the DevOps organization created their own process to allow projects to set up an inner source infrastructure. The new process helps in using and connecting already existing global tools to form a complete inner source infrastructure for a project including repository hosting, a community platform and build servers. The process for inner source infrastructure is currently limited to the organization owning the DevOps organization. Although this enclosing organization is large, it is not currently on the global corporate level. Trying to get this process adopted also as a company global process was mentioned as a future goal.

Discussion

The lack of a global inner source program is once again brought up, as mentioned previously in the discussions regarding both license considerations (see 4.3.3 RQ2 discussion) and discoverability (see 4.4.5). However, this time it is seen that an inner source program might be beneficial also from a more technical perspective. Namely that inner source project creation is difficult for reasons such as firewalls and other network security considerations. The process that was created by the DevOps organisation closely matches the infrastructure-based inner source model Gurbani et al. (2010) in which technical infrastructure is provided for the projects. However, as in the infrastructure-based model, the support from the DevOps organization is limited to providing the technical foundation, the projects are otherwise left mostly to themselves. It might be worth considering if this is sufficient support for the inner source projects, or if additional resources should be provided. For example, if the projects that are made inner source are often a prime technology of the company, then the project-specific model might prove better suited Gurbani et al. (2010).

5

Discussion

In this section, we discuss the potential threats to validity and gives recommendations on future work. Note that discussions on the results for the research questions are given in the respective results section.

5.1 Threats to Validity

In this section, potential threats to the validity of the study are presented. The framework employed is the one proposed by Runeson & Höst (2009), which covers four different types of validity: construct validity, internal validity, external validity, and reliability.

5.1.1 Construct Validity

Bjarnason et al. (2014) define construct validity as "how well the chosen research method has captured the concepts under study", and refers to how suited the methodology is for the task. Steps taken to mitigate this threat are outlined in detail in 3.3 Methodology Motivation, though a few points can be brought up to exemplify the efforts. First, interview subjects were ensured full anonymity and confidentiality of their answers. This has the effect of establishing a climate of trust in which the interviewees are comfortable in giving their sincere opinion on things. Furthermore, it helps in mitigating reactive bias, i.e. that participants should not be afraid of facing repercussions from e.g. management for statements made in an interview. Second, the interview guide was validated with a researcher at Chalmers University of Technology to make sure that relevant questions were asked and that the guide was not constructed in a way that was not in line with best practices. Third, interviews were if possible conducted in the interview subject's native language in an attempt to reduce misunderstandings; else it was verified that the interviewee was proficient enough in English as to not cause misunderstandings due to language barriers.

Another issue to do with construct validity that was discovered late in the process

is that of interview subject selection in regards to RQ2 on license issues with third party products. It could have been a good idea to complement the conducted project member interview with interviews with legal experts in the dedicated third party products organization within Ericsson. Such interviews would have laid a sounder legal base and would likely have strengthened our conclusions for the second research question. Furthermore, documentation on processes that regards third party product inclusion could, in addition to the interview data, have been acquired to further strengthen the findings. As mentioned in 3.4.2 Data Collection: Semi-structured Interviews, we attempted to mitigate this threat by interviewing project members that we suspected had some knowledge of this aspect.

5.1.2 Internal Validity

If conclusions are drawn in which not all factors are taken into account, or some factors that may influence the outcome are unknown, then there may be a threat to internal validity present (Runeson & Höst, 2009). Although this thesis attempts to draw some conclusions of why things are the way they are, it is not the primary objective of the thesis; as detailed in 3.3 Methodology Motivation, the objective is exploratory rather than explanatory. Nevertheless, one type of internal validity that can be discussed is that of researcher bias. We believe that the bias in this study is low, mainly due to being carried out by two researchers rather than one and being supervised by a university researcher with whom ideas were discussed, iterated on, and refined. Moreover, findings were discussed with the aforementioned researcher and submitted to peer review seminars conducted at the university in further efforts to take in more viewpoints and ensure that those of the researchers were not the only ones considered.

5.1.3 External Validity

External validity refers to the extent which it is possible to generalize the findings of the study (Runeson & Höst, 2009). Runeson & Höst (2009) argue that for a case study "the intention is to enable analytical generalization where the results are extended to cases which have common characteristics and hence for which the findings are relevant", which should be taken to mean that the findings of a case study can be generalized to similar cases. This notion is taken into account by in detail describing the case company and the cases inside the company which the case study aimed to study. Furthermore, the classification framework by Capraro & Riehle (2016) aids in the generalizability of the results. In the consideration of whether the findings in this study can be applied to a given inner source project, the inner source project in question can be compared to the classification of the study's cases. If the classification of the inner source project matches one of the cases, the findings for that case may be more applicable due to that similarity.

Another issue to take into account is that the study was only carried out at Ericsson. Attempts were made to validate the findings with similar companies but ultimately were not successful. This fact limits the study in that the findings are sensitive to cultural, company political, regional, and managerial factors. Furthermore, only two cases were studied which also hinders the generalizability of the results. Also here attempts were made to find more suitable cases, but these too were unsuccessful. More cases studied would have provided opportunities to contrast the findings between the cases to a greater extent and increased the validity of the results.

5.1.4 Reliability

This type of validity is about ensuring that if another researcher attempted to conduct the same study using the provided methodology, the results should be similar or even the same (Runeson & Höst, 2009). Reliability was addressed by meticulously describing the methodology, including an anonymized interviewee list, the interview process, the way in which thematic analysis was conducted, and by including the interview guides (both the English and Swedish versions, see Appendix A) that were used through all interviews in the case study.

5.2 Future Work

This thesis contributes to the field of inner source research by identifying three aspects that are especially important when applying inner source development practices to product projects, i.e. projects that have an external customer as the end receiver of the software. These three aspects have not been previously brought up in literature and were in this thesis explored by performing a case study on two large inner source projects at Ericsson. The aspects can be used in future research in the inner source field. We recognize the following areas where future research could be conducted.

Validate findings with product project
While the findings in this thesis sought to explore aspects of inner source development that may be interesting from the viewpoint of a product project, it does not validate these findings with an actual product project. It could, therefore, be interesting to study how some of the findings presented in this thesis might be implemented in a product project and how well these findings translate from an internal inner source project to a product project.

Validate findings with another company
As mentioned in the discussion to the inner source project implications found as part of RQ2, we believe that some of these implications are specific to the case company that was part of this study. As such, it would be interesting to validate this belief

with another company in order to investigate if this is the case, and if so to what extent the implications are shared or not shared between multiple companies.

Classify additional inner source projects using the framework by Capraro & Riehle (2016)
The classification framework by Capraro & Riehle (2016) that was described and used in this thesis proved useful both in the understanding of the studied cases and in the analysis and the activity of contrasting the aforementioned cases. In the light of these experiences, it may be of value to classify more inner source projects using the framework, as well as to study the prevalence of each of the classes. Such studies may provide value both to the field of inner source research and to the studies themselves if additional analysis is performed in conjunction with the classification. This same sentiment is repeated by the original authors of the framework.

6

Conclusion

In this thesis, we have presented the analysis of a case study which explored two cases at Ericsson. This analysis did not reach any generally applicable best practices. However, the findings from the research questions clearly indicate good practices in prioritization, license issues with third party products, and how to market an inner source project inside a company. The findings of these aspects are summarized below.

For prioritization of features, according to the definition given in the problem statement, it can be seen that this is performed in three steps: elicitation, delegating work, and actually implementing the feature. Elicitation is performed either by the core team coming up with features or the community requesting them. Feature requests from the community can come from for example forums, stakeholder boards or by talking directly to the core team on site. The step of delegating work is where it is decided if the core team or the community should implement the elicited features. The core team often performs the implementation if it is seen, for example, that the feature will benefit many stakeholders or if the feature is too complicated to be feasible for the community to implement. The community implements the feature if it is too niche to be of use to other stakeholders, or if the community member is experienced enough to handle the implementation him or herself. When it comes to implementation, a notable practice to facilitate contributions from the community is that of the 1/3-review, in which contributors are urged to contact the core team once a code skeleton and a rough sketch of the system has been drawn up in order to validate current progress. When it is decided that the core team should implement a feature, it can be observed that the feature is prioritized loosely according to agile practices.

Two themes are found for the aspect of the license issues related to the usage of third party products. First, it can be seen that there are certain implications for inner source projects when it comes to third party products and the legal considerations required for their associated licenses. Specifically, inner source projects are not as strictly required to handle license issues as a product project would be[1]. This is assumed to be specific to the case company, but it could not be verified within the scope of the study. Second, it can be observed that the two projects take steps to

[1]Ericsson have the procedures in place to fully follow all legal requirements.

make sure that third party dependencies are handled carefully if included. Code reviews and audits are performed in order to control what third party components are used in the inner source projects. The preferred strategy for third party product handling differs between the two projects. In the UI SDK, the strategy is to simply avoid such dependencies in the first place, and rather rely on internally written code. In JCAT however, the opposite is the case; first check what is available as third party products, and then implement any remaining functionality that is needed.

In terms of discoverability, four aspects were discovered that aid or could be of use in marketing an inner source project: project finds people, people find project, top-level support, and inner source program. It can be seen that in the infancy of the studied cases, it was to a greater extent the core teams who took on the responsibility of spreading awareness of the projects, for example through hackathons, giving demos on site, and conducting presentations. As the projects matured and the communities around them grew, more marketing responsibility could be delegated to the community members who spread the word to their peers. Support from upper management was brought up as a key point in both studied cases, both of whom have been recommended as the preferred tool within their respective domains. Another facet of top-level support is that of correlation between project budget and the number of users. Such a correlation is found in neither of the projects despite its existence being reasonable to assume, especially for JCAT which is classified to be owned by all organizational units. There also seems to be a lack of a central inner source platform where inner source projects and their related community pages, such as forums, can be hosted and where it is also possible to explore which inner source projects are available in the organization.

Bibliography

Asay, M. (2007). Microsoft office experiments with open source (development). [Online; accessed April 12th 2017].

Asundi, J. (2001). Software engineering lessons from open source projects. In *Proceedings of the 1st Workshop on Open Source Software Engineering, Toronto, ON, Canada*.

Bjarnason, E., Runeson, P., Borg, M., Unterkalmsteiner, M., Engström, E., Regnell, B., Sabaliauskaite, G., Loconsole, A., Gorschek, T., & Feldt, R. (2014). Challenges and practices in aligning requirements with verification and validation: a case study of six companies. *Empirical Software Engineering*, *19*(6), 1809–1855.

Braun, V. & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative Research in Psychology*, *3*(2), 77–101.

Capraro, M. & Riehle, D. (2016). Inner source definition, benefits, and challenges. *ACM Computing Surveys (CSUR)*, *49*(4), 1–36.

Dinkelacker, J., Garg, P. K., Miller, R., & Nelson, D. (2002). Progressive open source. In *Proceedings of the 24th International Conference on Software Engineering*, (pp. 177–184). ACM.

Ericsson (2017). Company facts. [Online; accessed April 13th 2017].

Fitzgerald, B. (2011). Open source software: Lessons from and for software engineering. *Computer*, *44*(10), 25–30.

Fortune (2016). The fortune 2016 global 500. [Online; accessed May 5th 2017].

GitHub (2016). The state of the octoverse 2016. [Online; accessed April 12th 2017].

Gurbani, V. K., Garvert, A., & Herbsleb, J. D. (2006). A case study of a corporate open source development model. In *Proceedings of the 28th international conference on Software engineering*, (pp. 472–481)., 1134352. ACM.

Gurbani, V. K., Garvert, A., & Herbsleb, J. D. (2010). Managing a corporate open source software asset. *Commun. ACM*, *53*(2), 155–159.

Hove, S. E. & Anda, B. (2005). Experiences from conducting semi-structured interviews in empirical software engineering research. In *11th IEEE International Software Metrics Symposium (METRICS'05)*, (pp. 10 pp.–23).

Lindman, J., Riepula, M., Rossi, M., & Marttiin, P. (2013). Open source technology in intra-organisational software development—private markets or local libraries. In J. S. Z. Eriksson Lundström, M. Wiberg, S. Hrastinski, M. Edenius, & P. J. Ågerfalk (Eds.), *Managing Open Innovation Technologies* (1 ed.). (pp. 107–121). Berlin, Heidelberg: Springer Berlin Heidelberg.

Lindman, J., Rossi, M., & Marttiin, P. (2010). Open source technology changes intra-organizational systems development-a tale of two companies. In *European Conference on Information Systems*.

Martin, G. & Lippold, A. (2011). Forge.mil: A case study for utilizing open source methodologies inside of government. In S. A. Hissam, B. Russo, M. G. de Mendonça Neto, & F. Kon (Eds.), *Open Source Systems: Grounding Research: 7th IFIP WG 2.13 International Conference, OSS 2011, Salvador, Brazil, October 6-7, 2011. Proceedings* (pp. 334–337). Berlin, Heidelberg: Springer Berlin Heidelberg.

Melian, C. (2007). *Progressive open source the construction of a development project at Hewlett-Packard*. Economic Research Institute, Stockholm School of Economics (EFI).

Mockus, A., Fielding, R. T., & Herbsleb, J. D. (2002). Two case studies of open source software development: Apache and mozilla. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *11*(3), 309–346.

Nakakoji, K., Yamamoto, Y., Nishinaka, Y., Kishida, K., & Ye, Y. (2002). Evolution patterns of open-source software systems and communities. In *Proceedings of the International Workshop on Principles of Software Evolution*, IWPSE '02, (pp. 76–85)., New York, NY, USA. ACM.

Perens, B. (1999). The open source definition. *Open sources: voices from the open source revolution*, *1*, 171–188.

Raymond, E. (1999). The cathedral and the bazaar. *Knowledge, Technology & Policy*, *12*(3), 23–49.

Riehle, D., Ellenberger, J., Menahem, T., Mikhailovski, B., Natchetoi, Y., Naveh, B., & Odenwald, T. (2009). Open collaboration within corporations using software forges. *IEEE software*, *26*(2), 52–58.

Robles, G. (2004). A software engineering approach to libre software. *Open Source Jahrbuch*, *2004*.

Robson, C. (2002). *Real world research* (2nd ed.). Blackwell Publishing.

Runeson, P. & Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, *14*(2), 131–164.

Seale, C., Gobo, G., Gubrium, J. F., & Silverman, D. (2004). *Qualitative research practice*. Sage.

Seaman, C. B. (1999). Qualitative methods in empirical studies of software engineering. *IEEE Transactions on software engineering*, *25*(4), 557–572.

Stol, K.-J., Avgeriou, P., Babar, M. A., Lucas, Y., & Fitzgerald, B. (2014). Key factors for adopting inner source. *ACM Transactions on Software Engineering and Methodology*, *23*(2), 1–35.

Taylor, S. J., Bogdan, R., & DeVault, M. L. (2016). *Introduction to qualitative research methods: a guidebook and resource*, volume 4th. Hoboken, New Jersey: John Wiley & Sons.

Torkar, R., Minoves, P., & Garrigós, J. (2011). Adopting free/libre/open source software practices, techniques and methods for industrial use. *Journal of the Association of Information Systems*, *12*(1), 88–122.

van der Linden, F. (2009). Applying open source software principles in product lines. *Upgrade*, *10*(2), 32–41.

Vitharana, P., King, J., & Chapman, H. S. (2010). Impact of internal open source development on reuse: Participatory reuse in action. *Journal of Management Information Systems*, *27*(2), 277–304.

Wesselius, J. (2008). The bazaar inside the cathedral: Business models for internal markets. *IEEE Software*, *25*(3), 60–66.

Whittaker, J. A., Arbon, J., & Carollo, J. (2012). *How Google tests software*. Addison-Wesley.

Wikipedia (2017). List of the largest information technology companies. [Online; accessed May 5th 2017].

A

Appendix 1

A.1 English Interview Guide

Pre-Interview:

- Our study
  - Inner source at Ericsson (with Sima Nordlund)
  - Investigate three especially interesting aspects of inner source by interviewing people in inner source projects at Ericsson
    - The three are: prioritization, legal issues (licenses) and discoverability
- Explain that your answers will be confidential, you will be anonymous
- Is it okay if we record this interview?
  - Transcribed, used as data for the study

Intro:

- What is your role and title, and what do they entail?
- How long have you been at your role and how long at Ericsson?
- What is your experience with open source and/or inner source?

RQ1. How are features prioritized in inner source projects?

- Who are the stakeholders of the project?
  - Core Team?
- How is the direction in which to steer the project decided?

  - Stakeholder meeting, roadmap

- How are feature requests prioritized?
  - Difference between “core team” and non-”core team”?
- What are the inclusion criteria for new contributions?
  - Feature bloat (i.e. non-generic functionality, features not in line with project vision)
  - Contributions planned in advance, or evaluated when implemented

RQ2. How are legal issues related to the usage of third party products handled in inner source projects?

- How is it decided which third party products or libraries are allowed inside the software, in the sense that it doesn't contain restrictive licenses?
  - OSS license compatibility
  - Trade compliance
  - Ericsson Bazaar
- Who is responsible for ensuring only allowed third party products or libraries end up in the software?
  - If lieutenants: How they are chosen?
  - Who is responsible if it should happen anyway?
- How does the vetting process work concretely? (e.g. I want to contribute changes, how are they verified not to violate any licenses/trade compliance/etc.)
  - Code review by “trusted member”
  - Continuous Integration test

RQ3. How can an inner source project spread awareness of its existence within the company where it is developed?

- How important is getting new contributors and stakeholders to this project?
- What is the project's strategy for making employees aware of its existence?
  - What is the difference between finding new users and new contributors?
- Are there any incentives from “above” to find additional users and/or contributors?

  - Money from stakeholders?

- What could be improved in order to make more employees aware of the project?

Outro

- Is there anything that we forgot to ask? Anything else you think we should know?
- Anyone else we should talk to?

A.2 Swedish Interview Guide

Pre-Interview:

- Our study
  - Inner source på Ericsson (med Sima Nordlund)
  - Undersöka tre speciellt intressant aspekter av inner source genom att intervjua folk med i inner source-projekt på Ericsson
    * De tre är: prioritering, juridiska aspekter (licenser) och discoverability
- Förklara att dina svar kommer vara konfidentiella, du kommer vara anonym
- Är det okej om vi spelar in den här intervjun?
  - Transkriberad, används som data i studien

Intro:

- Vilken är din roll och titel, och vilka uppgifter har du?
- Hur länge har du haft den här rollen och hur länge har du varit på Ericsson?
- Vad är din erfarenhet med open source och/eller inner source?

RQ1. How are features prioritized in inner source projects?

- Vilka är projektets stakeholders?
  - Core team?
- Hur bestäms det i vilken riktning projektet ska tas?
  - Stakeholdermöte, roadmap

- Hur prioriteras feature-förfrågningar?
  - Skillnad mellan “core team” och non-“core team”?
- Vad har ni för kriterier för att ta med nya contributions?
  - Feature bloat (dvs. icke-generisk funktionalitet, features som inte är i linje med visionen)
  - Contributions som planerats i förväg, eller evaluerad när det implementeras

RQ2. How are legal issues related to the usage of third party products handled in inner source projects?

- Hur bestäms det vilka tredjepartsprodukter eller bibliotek som ska tillåtas i produkten, när det gäller att de inte ska innehålla restriktiva tredjepartslicenser?
  - OSS-licens kompatibilitet
  - Trade compliance
  - Ericsson bazaar
- Vem är ansvarig för att se till att tredjepartsprodukter eller bibliotek inte kommer in i produkten?
  - Om “trusted lieutenants”: hur väljs de?
  - Vem är ansvarig om det skulle hända ändå?
- Hur fungerar urvalsprocessen rent konkret? (t.ex. Jag vill bidra med någon förändring, hur verifieras det att de inte bryter mot licenser osv?)
  - Code review av “trusted member”
  - Continuous integration test

RQ3. How can an inner source project spread awareness of its existence within the company where it is developed?

- Hur viktigt är det att få in nya utvecklare och användare i detta projekt?
- Vad är detta projektets strategi för att uppmärksamma anställda om dess existens?
  - Vad är skillnaden mellan att hitta nya användare och nya utvecklare?
- Finns det några incitament “uppifrån för att få in nya användare eller utvecklare?

  - Pengar från stakeholders?

- Vad kan förbättras för att uppmärksamma fler personer på att projektet finns?

Outro

- Har vi glömt att fråga något? Något annat du tycker att vi borde veta?
- Någon annan vi borde prata med?