What to share, when, and where: balancing the objectives and complexities of open source software contributions

Johan Linåker[1] · Björn Regnell[1]

Abstract

Context: Software-intensive organizations' rationale for sharing Open Source Software (OSS) may be driven by both idealistic, strategic and commercial objectives, and include both monetary as well as non-monetary benefits. To gain the potential benefits, an organization may need to consider what they share and how, while taking into account risks, costs and other complexities.

Objective: This study aims to empirically investigate objectives and complexities organizations need to consider and balance between when deciding on what software to share as OSS, when to share it, and whether to create a new or contribute to an existing community.

Method: A multiple-case study of three case organizations was conducted in two research cycles, with data gathered from interviews with 20 practitioners from these organizations. The data was analyzed qualitatively in an inductive and iterative coding process.

Results: 12 contribution objectives and 15 contribution complexities were found. Objectives include opportunities for improving reputation, managing suppliers, managing partners and competitors, and exploiting externally available knowledge and resources. Complexities include risk of loosing control, risk of giving away competitive advantage, risk of creating negative exposure, costs of contributing, and the possibility and need to contribute to an existing or new community.

Conclusions: Cross-case analysis and interview validation show that the identified objectives and complexities offer organizations a possibility to reflect on and adapt their contribution strategies based on their specific contexts and business goals.

Keywords Open source software · Software product management · Requirements engineering · Contribution strategy · Community strategy

✉ Johan Linåker
johan.linaker@cs.lth.se

Extended author information available on the last page of the article.

1 Introduction

Sharing, or contributing, software artifacts (e.g., features, projects, and frameworks) as Open Source Software (OSS) is a common practice among software-intensive organizations today (Munir et al. 2016). By "opening up" (Chesbrough et al. 2014), an organization can exploit the external workforce residing in the OSS communities that develops and maintains the OSS. Improved product innovation, accelerated development, lower maintenance cost, as well as improved branding and reputation, are some of the potential benefits that may motivate (Munir et al. 2016, 2018a; Stuermer et al. 2009; Henkel 2006; Lindman et al. 2009). The motive may also be driven by pure idealism and being a good OSS citizen (Jansen et al. 2012). For some organizations, OSS may have a more direct connection to the business model or strategy, e.g., as a basis for complementary products and services (Shahrivar et al. 2018; Andersen-Gott et al. 2012; Spijkerman and Jansen 2018), or as a means to create a new standard or compete with existing ones (Lindman et al. 2009; Jansen et al. 2012; Henkel 2006; West 2003). Objectives for why an organization would choose to contribute software artifacts as OSS does, however, not have to be limited to one or the other (Andersen-Gott et al. 2012).

In this paper, we present the results from an empirical study on organizations' rationale for sharing software artifacts as open source. In the context of this study, we introduce the term **Contribution Objective** (CO), which we define as *a purpose for contributing a software artifact, motivated by a monetary or non-monetary benefit that is enabled or resulted directly or indirectly as a consequence of the contribution.*

To gain potential benefits by acting in line with contribution objectives, an organization needs to access the external workforce, either by contributing its software artifacts to an existing OSS community or by creating a new community, each with its respective costs and risks (Dahlander and Magnusson 2008). In either case, the organization then needs to work actively to align its internal strategy with the community where they are a stakeholder among many, potentially including competitors with conflicting agendas (Dahlander and Magnusson 2005, 2006). An organization, therefore, may have to consider not just *where*, but also *what* it contributes, and *when*. Risks include giving away differentiating functionality (Wnuk et al. 2012; Van der Linden et al. 2009; Henkel 2006, 2008; West and Gallagher 2006; Iivari et al. 2008), or contributing too late and having to choose between adopting the alternative solution or maintaining an internal solution alone (Linåker et al. 2018; Wnuk et al. 2012). Hence, there are several potential costs and risks tied to a contribution.

In the context of this study, we refer to these potential costs and risks as **Contribution Complexities** (CCs) and define them as *aspects related to a software artifact that may complicate the contribution of the artifact, or imply a cost or risk as a result thereof, either directly or indirectly.* As with contribution objectives, not all complexities may be relevant for all organizations (Linåker et al. 2018).

By not considering relevant contribution objectives or complexities, an organization may risk making a contribution that could be damaging, or inadvertently block a contribution that could have been beneficial for the organization and its business goals (Wnuk et al. 2012). To gain the expected benefits, organizations therefore need to link their business goals with their decisions on what they contribute as OSS. Despite the problematic context, research on how organizations can develop and use such strategies is limited (Munir et al. 2016), with some exceptions (Wnuk et al. 2012; Weikert and Riehle 2013; Linåker et al. 2018). This leads us to define the following research questions:

RQ1   What contribution *objectives* should a software-intensive organization consider when assessing if, where and when a software artifact should be shared as OSS?

RQ2   What contribution *complexities* should a software-intensive organization consider when assessing if, where and when a software artifact should be shared as OSS?

We addressed these research questions by conducting a multiple-case study at three software-intensive organizations with an iterative approach spanning over two research cycles. Based on an inductive coding of twenty semi-structured interviews divided among the three organizations, 12 contribution objectives and 15 contribution complexities were identified.

An organization may, based on the contribution objectives and complexities that they find relevant for their context and business goals, make informed decisions on what software artifacts that should be released as OSS, when and where. For a specific artifact, the question "*what?*" regards if the artifact should be contributed in full or kept closed, or if certain parts can be contributed under certain conditions. The question "*when?*" refers to when in time an artifact should be contributed. Finally, the question "*where?*" asks whether the artifact should be contributed to one of many existing OSS communities or if a new community should be established. Answers to these questions are input to a *contribution strategy* for the software artifact under consideration. The overall objective of the work presented in this paper is to elicit such answers from real-world example cases, and start building a relevant list of considerations that can help when developing a contribution strategy.

The rest of this paper is structured as follows. Sections 2 presents related and previous work to this study. Section 3 presents the research design and background information to the three case organizations. Section 4 presents the identified contribution objectives and complexities, and Section 5 provides a discussion on the objectives and complexities in regards to related work. Section 6 presents a discussion on threats to validity, while Section 7 concludes the paper. Appendices A–D provide detailed information about interview instruments and findings.

2 Related and Previous Work

This section provides an overview of related work to the two concepts of Contribution Objectives and Contribution Complexities. This is followed by an overview of contribution strategy research after which we present a summary of the related and previous work.

2.1 Benefits of Sharing Software as OSS

Several studies have systematically surveyed the literature and to different extents covered the benefits for why software should be shared as OSS (Shahrivar et al. 2018; Munir et al. 2016; Höst and Orucevic-Alagic 2011; Hauge et al. 2010). We categorize the benefits into four different themes.

A common theme is the cost-saving aspects (Munir et al. 2018b, 2016; Andersen-Gott et al. 2012). By extending the resource-base (Dahlander and Magnusson 2008) and agreeing on a common standard (West and Gallagher 2006), organizations can share the maintenance and quality assurance, accelerate the development and potentially decrease their time-to-release and market (Munir et al. 2018a; Stuermer et al. 2009; Henkel 2006; Lindman et al. 2009; Olsson and Bosch 2017; Lundell et al. 2011). By freeing up internal resources, they can focus on more value-adding activities (Munir et al. 2018a; Van der Linden et al. 2009;

Lindman et al. 2009). On the opposite, by adopting a less symbiotic relationship to the OSS community (Dahlander and Magnusson 2005), an organization will have to maintain an internal branch of the OSS project, which may become costly depending on the number of modifications that need to be applied to new releases of the OSS project (Wnuk et al. 2012; Ven and Mannaert 2008). Hence, to attain these potential benefits, active engagement and a symbiotic relationship may be needed with the OSS community (Chengalur-Smith et al. 2010; Dahlander and Wallin 2006).

Another common theme is the innovation aspects (Munir et al. 2016; Andersen-Gott et al. 2012), which can be both product and process-oriented (Munir et al. 2018b). By opening up the innovation process (Chesbrough and Appleyard 2007) and "pooling" the R&D/product development (West and Gallagher 2006), organizations get access to an external workforce (Lundell et al. 2010), which may bring increased knowledge sharing (Nagle 2018; Lundell et al. 2011) and innovation at a lower cost (Stuermer et al. 2009; Ziegler et al. 2014). However, this external workforce should be seen as a complement rather than a substitute for internal knowledge and development (Stam 2009; Dahlander and Magnusson 2008). Munir et al. (2016) describe it as a catalyst for ideas that may help organizations in broadening their offerings. Hence, an organization may question how much of its internal R&D and innovation process it should outsource to a community (ÅGerfalk and Fitzgerald 2008).

A third theme can be tied to improved reputation (Munir et al. 2016; Ven and Mannaert 2008). By creating a community or contributing to an existing community, an organization can create a marketing channel both towards (potential) customers, as well as future employees and have a positive effect on internal developers' satisfaction (Lundell et al. 2010; Dahlander and Magnusson 2008; Riehle 2011; Stuermer et al. 2009; Henkel 2006). The improved reputation can turn into a competitive advantage (Henkel et al. 2014) and legitimize the use of the OSS from a public perspective (Dahlander and Wallin 2006). An organization's customers are offered an opportunity to avoid vendor-lockin, and the ability to customize the software to internal needs (Lundell et al. 2011; Munir et al. 2018a).

A fourth theme concerns control aspects (Linåker et al. 2018). If an OSS community has a meritocratic coordination process in place (Shaikh and Henfridsson 2017), influence on the development direction of the community may be gained by participating in the development and maintaining a symbiotic relationship (Linåker et al. 2019a; Dahlander and Magnusson 2005, 2006; Butler et al. 2018; Schaarschmidt et al. 2015; Syeed et al. 2017; Nguyen-Duc et al. 2019). This may help steer the community including competitors and to manage potentially conflicting agendas (Linåker et al. 2019a; West and Wood 2008; Munir et al. 2016; Schaarschmidt et al. 2015; Mäenpää et al. 2018).

Influence may also come implicitly when an organization's project or a feature is released and accepted as a standard solution, either within an existing community or as a new community (Munir et al. 2018a). If contributed to an existing community, other organizations will either have to accept and adapt, maintain internal forks of their solutions, or attempt to contribute their solutions in competition with the solution already established within a community (Linåker et al. 2018). If released as a new community and traction is gained, it can potentially become a new standard or compete with existing (Lindman et al. 2009; Jansen et al. 2012; Henkel 2006; West 2003), and create a surrounding ecosystem with complements from other organizations (West 2007; Hartmann and Bosch 2016).

Some of these themes may be more or less important depending on the type of organization. Munir et al. (2018b) present a theory of openness that categorizes organizations using OSS in their tools and infrastructure setups based on why and when they adopt and

share software as OSS. The "why" is either focused on reducing product development costs or building a symbiotic relationship with the OSS communities (Dahlander and Magnusson 2005). The "when" concerns whether a reactive or proactive strategy is adopted. In the former, an organization adapts to existing and upcoming OSS communities without taking any initiatives. In the latter, an organization has a long-term agenda and adapts its OSS community engagements or create new communities accordingly.

2.2 Costs and Risks of Sharing Software as OSS

Deciding what should be contributed is a complex matter (Munir et al. 2016). Even though the amount of a software that may be considered differentiating is often limited (Lindman et al. 2009; Van der Linden et al. 2009), the risk of sharing differentiating functionality and sensitive Intellectual Property Rights (IPR), and as a consequence losing a competitive edge, is a recognized challenge in literature (e.g., Wnuk et al. 2012; Van der Linden et al. 2009; Henkel 2006, 2008; West and Gallagher 2006; Iivari et al. 2008). Instead of disqualifying a complete software artifact however, one approach may be to selectively reveal commodity or enabling parts while keeping differentiating parts closed (West 2003; Henkel 2006; Stuermer et al. 2009). An alternative approach may be to "spinout" (West and Gallagher 2006) disclose the software artifact under a restrictive copy-left license (West 2003), e.g., the General Public Licence version 2 and 3, or the Affero GPL.[1] This is a common approach for commercial OSS organizations (Riehle 2012) using a dual-license approach to appropriate and capture value from customers (Chesbrough and Appleyard 2007). Combinations can be found, e.g., where a core OSS project is permissively licensed, while certain extensions or improvements are licensed with more restrictive licenses (Deodhar et al. 2012), or kept proprietary (Weikert and Riehle 2013).

A related challenge is determining when software should be contributed (Wnuk et al. 2012). Several studies have attempted to model and identify an optimal timing (Haruvy et al. 2008; Kort and Zaccour 2011; Caulkins et al. 2013). Caulkins et al. (2013) for example, identify costs related to the development and adapting the business model, along with the software quality as factors affecting when software should be released as OSS. Quality is in this case not just referred to the number of errors, but to the software's features and functionality. The shift in how quality changes can be compared to where in the commoditization process a software artifact is. I.e., the *"software artifact's value depreciation and how it moves between a differential to a commodity state, i.e., to what extent the artifact is considered to help distinguish the focal organization's product offering relative to its competitors"* (Linåker et al. 2018). As suggested by Van der Linden et al. (2009), a first step may be to open up the software in joint ventures or closed strategic alliances (Linåker et al. 2018), while a third step may be to release it as OSS. Wnuk et al. (2012) describe the challenge as a balance between losing a competitive edge and increased maintenance costs.

Where to contribute is a third challenge. When contributing the software artifact to an existing OSS community not governed by the organization itself (O'Mahony 2007), the organization needs to consider the requirements engineering process of the community (Scacchi 2002). In this context, the organization is a stakeholder among many and needs to consider potentially conflicting agendas from other stakeholders (Munir et al. 2016; Schaarschmidt et al. 2015; Mäenpää et al. 2018; Linåker et al. 2019b). If there is a misalignment between the organization's and the community's Requirements Engineering

---

[1]https://opensource.org/licenses/alphabetical

(RE) process (Ernst and Murphy 2012; Alspaugh and Scacchi 2013; Kuriakose and Parsons 2015), the organization may need to influence the development direction of the community (Munir et al. 2018a). If the organization lacks the influence needed, they need to consider the cost of gaining it (Linåker et al. 2019a). An option is to release the software as an independent OSS project and build a new community around it, which may require significant investment as well (Kilamo et al. 2012; West and O'Mahony 2005; Dahlander and Magnusson 2008).

2.3 Contribution Strategies

Wnuk et al. (2012) highlighted the importance of contribution strategies early on. Research on the topic has however been limited (Munir et al. 2016) with some exceptions (Wnuk et al. 2012; Weikert and Riehle 2013; Linåker et al. 2018).

In previous work (Linåker et al. 2018), the first author of this study conducted a case study on the contribution strategy decision process at Sony Mobile. Through the case study, a Contribution Acceptance Process (CAP) model was designed with the purpose to help organizations decide if a software artifact should be shared as OSS. A software artifact (e.g., features or projects) is valued according to its *business impact* (how much you profit from the component) and *control complexity* (how hard the technology and knowledge behind the artifact is to acquire and control). With the help of a series of questions provided, the software artifact could be ranked qualitatively and placed in a two-by-two matrix (see Fig. 1), with each cell representing a certain type of generic artifact with its own contribution strategy that is proposed for the artifact.

As an example, one of the four artifact types concerned strategic artifacts (upper right quadrant in Fig. 1) (Linåker et al. 2018). These artifacts have a high business impact and a high control complexity. They contain differentiating value and provides a competitive edge to the organization. Differentiating parts should be kept closed while enabling parts, should be contributed. The contribution is recommended to be made to a community where the organization has a high level of influence. If not possible, a new community may be created.
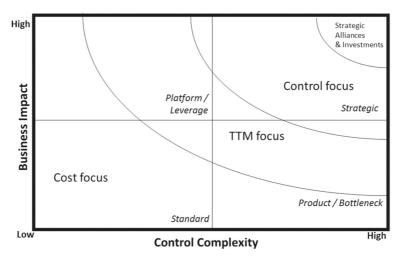

Fig. 1 The Contribution Acceptance Process (CAP) model as presented in Linåker et al. (2018). Based on how a software artifact is valued in regards to its business impact and control complexity, a contribution strategy is elicited pending the artifact's placement on the grid

The contribution strategy could then be fine-tuned based on a series of objectives (Linåker et al. 2018). The more strategic an artifact is (higher business impact and control complexity) the faster the artifact (or enabling parts of it) should be contributed to establishing the artifact as the standard solution. The organization could thereby avoid having to adapt to competing solutions and instead strive towards steering its competitors. For this reason, these kinds of artifacts should be prioritized before standard artifacts, which may be considered as standard knowledge.

An example of a strategic artifact included a multimedia framework that enabled differentiating camera functionality (Linåker et al. 2018). The framework was contributed, while the camera functionality could be kept closed. This gave Sony Mobile the advantage of not having to refactor or adapt to competing frameworks and still keep differentiating functionality closed.

The CAP model can be used both in a proactive and reactive approach (Linåker et al. 2018). In the proactive, it is used to map a series of software artifacts, e.g., features in the product planning process (Kittlaus and Fricker 2017). A cross-functional team may be assembled, including e.g., representatives from marketing, legal and product development. The team can iterate the mapping process comparing features against each other through consensus-seeking discussions. In the reactive approach, the CAP model is used in a similar manner but for making decisions in regard to incoming contribution requests from the organization's development teams.

The validation of the CAP model showed that the CAP model provided a good foundation for discussion. Feedback pointed out that the questions and scale used to value a software artifact in terms of business impact and control complexity, was found useful but in need of being tailored to the context where the CAP model is applied. A highlighted concern for future work was to avoid making the following versions of the CAP model more complex.

2.4 Summary

The benefits that may incentivize an organization to contribute its software as OSS are many (Shahrivar et al. 2018; Munir et al. 2016; Höst and Orucevic-Alagic 2011; Hauge et al. 2010), as are the potential costs and risks that may remove or outweigh the benefits (Munir et al. 2016; Linåker et al. 2015). To gain the expected benefits, an organization, therefore, needs to consider the contribution objectives and complexities relevant to their context, and make an informed decision on what they contribute, where, and when, in alignment with their business goals. Related work on how organizations can develop such strategies is limited (Munir et al. 2016), with some exceptions (Wnuk et al. 2012; Weikert and Riehle 2013), including our previous work with Sony Mobile and the CAP-model (Linåker et al. 2018). The validation of the CAP-model pointed to a need for more general and less complex approaches for creating contribution strategies. This study, therefore, aims to identify and define a set of contribution objectives and complexities from which organizations can choose those relevant and thereby create contribution strategies based on their specific contexts and business goals.

3 Research Design

To answer the research questions **RQ1** and **RQ2** as defined in Section 1, we employed a multiple-case study (Runeson et al. 2012) in an iterative approach with two research cycles.

The method offers a way for generating in-depth knowledge and understanding of a phenomena in how and why it occurs (Easterbrook et al. 2008). In our study, this regards an exploratory investigation of what Contribution Objectives (COs) and Contribution Complexities (CCs) that the case organizations consider in decisions on whether a software artifact should be released as OSS, when in time, and if it should be contributed to an existing community or if a new community should be created. Answers to these questions together form the contribution strategy for the software artifact. The decision of arriving at such strategies makes up our unit of analysis (Runeson et al. 2012).

In total three case organizations were studied, one in the first research cycle and two in the second. Through this approach, we could develop the first set of contribution objectives and complexities which could then be used as a foundation when entering the second research cycle. Below we first describe the case organizations. We then describe how the research was carried out through the two research cycles.

3.1 Case Organizations

Below we describe and provide context to each of the three case organizations studied, denoted CaseOrg1-3. Per organization, we present a general description, along with a more in-depth overview of their contribution process as well as examples of OSS projects, which they have released, or are active contributors to.

3.1.1 CaseOrg1

General description: CaseOrg1 is a US-based media and technology company providing video, high-speed internet, smart home and voice services. They have 1000+ employees and develop their own software to enable and deliver their services to the customers. Having been passive consumers of OSS before 2006, they became active contributors starting in 2006. Since then, they have released several OSS projects and are active contributors in several others, as well as members of a number of OSS foundations.

Contribution process: Internally, they have an Open Source Programs Office set up to develop and manage e.g., contribution and compliance processes, and community management. Their contribution process starts with a developer filling out a contribution request form concerning a software artifact (e.g., feature or project). If the artifact regards a smaller contribution (e.g., a bug fix, or documentation), or a contribution to an OSS community deemed generally as non-competitive, approval can be gained online by the manager and the Open Source program office. For more significant components, the contribution request is managed by an OSS advisory board, which has representatives from legal, IPR, development and business functions within the organization. The board then gives a final decision on how to proceed. CaseOrg2 also has what they refer to as *sandbox approval*, which allows a project and identified contributors to be approved to contribute to a project without coming back for each patch. This governance set-up and contribution process is similar to that of Sony Mobile (Linåker et al. 2018).

Example OSS projects: One example concerns an internally developed Linux distribution that is embedded in hardware devices shipped to customers, which enables the delivery of CaseOrg1's consumer-oriented services. The software was initially developed to replace a proprietary option and was later released as OSS under the governance of an industry consortium. Another example concerns an infrastructure project, which enables the delivery

of services related to secure and reliant internet-traffic to business-oriented customers. The project was released under the governance of a neutral community with an existing OSS foundation. A third example regards a DNS-as-a-Service project originally developed to increase internal operational efficiency. The decision process is thoroughly reported on in previous work (Linåker et al. 2018).

3.1.2 CaseOrg2

General description: CaseOrg2 is a European-based hardware electronics manufacturer serving both business and private customers. They have 1000+ employees and develop their own software and orchestrate a software ecosystem (Jansen et al. 2009) to enable and deliver their services to the customers. This study has focused on its Tools department which develops and maintains development tools and infrastructure projects used by the organization's product development teams. A majority of the tools and infrastructure projects are OSS-based with active engagement from the Tools department in their respective communities. The active engagement includes continuous contributions of features and plugins to existing OSS communities as well as the release of new OSS projects.

Contribution process: CaseOrg2 does not have a dedicated Open Source Programs Office. The organization has two contribution processes set up, one for their product development teams, and one for the Tools department. The latter is less strict than the former as Case-Org2 generally considers the projects developed within the Tools department to provide a limited competitive edge to the organization. The contribution process used within the Tools department is initialized by a developer filling out a contribution request form concerning a software artifact (e.g., feature or project). If the contribution request is intended for an existing community, the request is managed by the department manager. If the contribution request concerns the creation of a new OSS community, the request is managed by the business unit manager. If deemed necessary the concerned manager will consult relevant functions within the organization such as Legal and IPR departments.

Example OSS projects: The Tools department has contributed and maintains several plugins to the Jenkins[2] and Gerrit[3] OSS projects. Jenkins is a build-server and Gerrit is a code-review tool, both used in CaseOrg2's continuous integration tool-chain. The Tools department has recently also started to create new OSS projects that fall outside existing communities. One such project was developed internally to allow for the creation and use of shorter URLs to internal resources. These are maintained as standalone projects on CaseOrg2's Github[4] page.

3.1.3 CaseOrg3

General description: The Swedish Public Employment Service[5] makes up the third case organization. They are non-anonymous but are referenced to as CaseOrg3 for consistency. CaseOrg3 is a public sector agency in Sweden with the main goal to facilitate and enable

---

[2]https://jenkins.io/

[3]https://www.gerritcodereview.com/

[4]https://github.com/

[5]https://arbetsformedlingen.se/

matching between job-seekers and employers on the Swedish labor market. The organization has 10 000+ employees of which 600 are employed within their IT division. The focus of this study is on a department within the IT division which aims to create a platform[6] on which private actors can build complementary products and services for matching job-seekers and employers. The platform, consisting of OSS projects and Open Data sources, is intended to help CaseOrg3 to move from the role of being a service provider to become a service enabler and help the platform's ecosystem members to collaborate and co-create, potentially resulting in accelerated innovation and a more efficient job-matching on the Swedish labor market.

Contribution process: CaseOrg3 does not have a dedicated Open Source Programs Office or contribution process in place. The studied department prioritize the release of internal projects based on what they believe is of most value to the platform's ecosystem of private actors.

Example OSS projects: The studied department has released a number of OSS projects consisting of small components that can integrate into existing applications, stand-alone end-user applications, and developer-focused tools and utilities.[7] One project is a video-conference-tool used internally at CaseOrg3 to facilitate remote interviews for employers and job-seekers. Another example concerns a search engine that can be used to access and search among job listings in CaseOrg3's Open Data sources with job ads.

3.2 Research Cycle 1

In the first research cycle, we investigated the problem context through a first case study of CaseOrg1. Data was gathered through six semi-structured interviews with employees from different areas of the organization, see Table 1. A questionnaire (see Appendix A) was created based on findings from an earlier reported case study of the contribution strategy decision process at Sony Mobile (Linåker et al. 2018) conducted by the authors of this study.

Interviewees were selected in order to gain different and complementary perspectives on CaseOrg1's contribution strategy decision process. Each of the interviews was audio-recorded and transcribed. An anonymized interview summary was presented to I1 and I2 to verify interpretations and clarify any misunderstanding. The transcripts were then coded with an inductive approach and audit trails maintained (Runeson et al. 2012). Using an iterative and refining coding process (Runeson et al. 2012; Robson 2011), sentences and paragraphs from the interviews were given descriptive topic sentences which then merged together in common codes and sorted under the two categories contribution objectives and complexities, mapping to **RQ1** and **RQ2** respectively. This rendered in a first set with a total of 16 objectives and 12 complexities (see Table 3).

Below we provide an example coding from CaseOrg1, which later was conceptualized as Contribution Objective CO3 after the updated coding from the second research cycle:

---

[6]https://jobtechdev.se/

[7]https://jobtechdev.se/doc/samples/

Table 1 List of interviewees from CaseOrg1-3

ID	Case organization	Title	Employment
I1	CaseOrg1	Open Source Program Officer	3 years
I2	CaseOrg1	Community Manager	9 years
I3	CaseOrg1	Director of Software Development	9 years
I4	CaseOrg1	Director of Software Architecture	14 years
I5	CaseOrg1	Vice President - Standards	16 years
I6	CaseOrg1	Chief Software Architect	13 years
I7	CaseOrg2	Project Manager	1 year
I8	CaseOrg2	Senior Developer	4 years
I9	CaseOrg2	Junior Developer	1 year
I10	CaseOrg2	Department Manager	5 years
I11	CaseOrg2	Business Unit Manager	1 year
I12	CaseOrg3	External Consultant	3 years
I13	CaseOrg3	Chief Digital Officer	6 years
I14	CaseOrg3	Department Manager	6 years
I15	CaseOrg3	Product Owner	9 years
I16	CaseOrg3	Project Manager	2 years
I17	CaseOrg3	Community Manager	1 year
I18	CaseOrg3	Security Engineer	1 year
I19	CaseOrg3	Senior Developer	6 years
I20	CaseOrg3	Senior Developer	5 years

– **Code**: Developer satisfaction

  – **Quote by I5**: *"…I think a real reason goes to staffing engagements and retention. It is important to people and I think a positive employment characterization that you get to engage with open source communities and that the company does release something as open source projects. I think that's a big selling point that people are looking for in whom they want to work for as an engineer."*.

3.3 Research Cycle 2

In the second research cycle, our goal was to validate the contribution objectives and complexities identified in the previous cycle while continuing to explore the problem context and identify new ones. The interview questionnaire was therefore updated based on previous findings. To validate the questionnaire and gain further input into its revision, we studied *contribution request forms* collected from seven different software-intensive organizations (see Table 2). A contribution request form contains questions about software artifacts, and often guidelines for, and examples of what types of contributions the organization generally accepts. The form is often submitted by the engineer or team behind the concerned software artifact. The contribution request forms give an indication of what the organizations consider when deciding whether the software artifact should be contributed.

Table 2 List of organizations from where contribution request forms has been gathered

ID	Business	Market	Employees	Use of OSS
O1	Consumer electronics	Worldwide	4 000+	Infrastructure & Products
O2	Consumer electronics	Worldwide	100 000+	Infrastructure & Products
O3	Software products	Worldwide	30 000+	Infrastructure & Products
O4	Software products	Worldwide	100 000+	Infrastructure & Products
O5	Telecom	North America	1 000+	Infrastructure & Products
O6	Consumer electronics	Worldwide	1 000+	Infrastructure & Products
O7	Industry organization	North America	-	-

The revised questionnaire (see Appendix B) was then used in semi-structured interviews with five employees from CaseOrg2 and nine employees from CaseOrg3. As in the former cycle, interviewees were selected in order to gain different and complementary perspectives on two organizations' contribution strategy decision process. All interviews were audio-recorded and transcribed. Anonymized interview summaries were presented to I7 and I8 from CaseOrg2 and I15 from CaseOrg3.

Using the coding schema from the first cycle, transcripts from CaseOrg2 were first coded, which resulted in three new COs and one CC being added. In the following step, the new coding schema covering CaseOrg1-2 was applied to transcripts from CaseOrg3, resulting in one new CO being added, six COs merged into three. Three new CCs were added, of which two were former COs. Transcripts from all case organizations were then reiterated, and a final coding scheme assembled. This resulted in two COs being merged into one, two COs converted to CCs, and six CCs being merged into three. In Table 3, the evolution, and saturation of the COs and CCs throughout the coding process are presented.

Below we present a continuation of the example code *Developer satisfaction* provided in Section 3.2. After coding the transcripts from CaseOrg2-3, further support was identified for *Developer satisfaction*. This code was then consolidated with the new code *Hire talent*, which was also identified in transcripts from CaseOrg1 in the reiteration, finally rendering in contribution objective CO3.

Table 3 Evolution and saturation of COs and CCs throughout the coding process consisting of four steps. The first coding schema was based on transcripts from CaseOrg1. This was then applied and revised based on transcripts from CaseOrg2, and then CaseOrg3. The coding schema was then reiterated, resulting in the final set of COs and CCs

	1. CaseOrg1	2. CaseOrg2	3. CaseOrg3	4. Final coding
COs (New)	16	3	1	0
COs (Merged)	0	0	$6 \rightarrow 3$	$2 \rightarrow 1$
COs (Removed)	0	0	$2 \rightarrow CC$	$2 \rightarrow CC$
COs (Total)	16	19	15	12
CCs (New)	12	1	3	2
CCs (Merged)	0	0	0	$6 \rightarrow 3$
CCs (Removed)	0	0	0	0
CCs (Total)	12	13	16	15

– **Contribution Objective CO3**: Improve employer branding

    – **Code**: Developer satisfaction

        • **Quote by I5**: *"... I think a real reason goes to staffing engagements and retention. It is important to people and I think a positivexemployment characterization that you get to engage with open source communities and that the company does release something as open source projects. I think that's a big selling point that people are looking for in whom they want to work for as an engineer.".*

        • **Quote by I19**: *"We think it is fun and it is positive for our personal satisfaction to contribute back".*

    – **Code**: Hire talent

        • **Quote by I8**: *"And then as a result of that, we already see it in some areas, because of our involvement in open standards or open source communities, we're able to attract a different type of engineer, or a different higher level engineer or value-add that they can bring to the company as a result of our openness and engagement. And so it's sort of a natural reinforcing cycle.".*

        • **Quote by I7**: *"Show that we are a modern firm with a presence in open source and that we contribute and not just consume, which attracts a lot of great developers and becomes a channel for us to attract new employees.".*
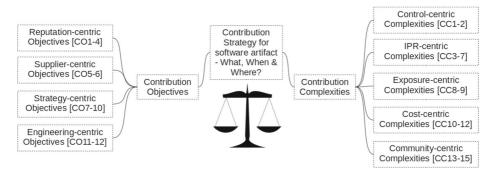
The revised set of COs and CCs were presented and discussed with I1 from CaseOrg1, I7 and I8 from CaseOrg2 and I15 from CaseOrg3. All interviews were audio-recorded and transcribed. The interviewees were first asked about the correctness of the COs and CCs that had been mapped to their organization. Interviewees were then asked if any COs or CCs not mapped to them were found relevant for the organization. Lastly, the interviewees were asked about the general completeness, applicability, and usability of the set of COs and CCs, and whether something was redundant, missing, or could potentially be modified. Existing COs and CCs were then refined based on interview findings, while none were added nor removed.

4 Results

In this section, we summarize the identified contribution considerations, grouped into Contribution Objectives (COs) and Contribution Complexities (CCs), which an organization may analyze and weigh against each other when deciding on a contribution strategy for a certain software artifact. In total, we present 12 COs and 15 CCs divided over four and five categories respectively (see Fig. 2).

All contribution considerations are maybe not relevant to all organizations and all software artifacts. To help guide decision-makers and those making contribution requests, the presented contribution considerations can help as input when forming an organization-specific contribution strategy. COs and CCs can then be described in the context of the focal organization and relevant questions asked in a contribution request form when setting up a contribution process within the organization. An individual intending to file a request is then

Fig. 2 This study presents 27 considerations (12 Contribution Objectives (CO) and 15 Contribution Complexities (CC)) that may need to be considered by an organization when deciding on a contribution strategy for a software artifact. The COs and CCs are divided into four and five categories respectively and are listed in Tables 4 and 5

enabled to beforehand understand the rationale used by the decision-makers when deciding on a contribution strategy, and thereby to provide motivated arguments why the request should be accepted. For further discussion, see Section 5.

The COs are presented, each with a number of (potential) key benefits, in Table 4, while the CCs are presented, each with a number of key concerns, in Table 5 and are further explained below. A detailed description of each consideration, together with example quotes from interviews, are given in Appendices C and D.

4.1 Contribution Objectives

In total, 12 COs were identified in the analysis process divided over four categories: reputation-centric objectives, supplier-centric objectives, strategy-centric objectives, and engineering-centric objectives. Below an overview is given to the COs in each category. A more detailed overview is presented in Appendix C where the COs are described separately with examples per identified benefit. Quotes from interviewees of the case organizations are given in Appendix C to provide context for each of the examples.

4.1.1 Reputation-centric Objectives

The reputation-centric objectives highlight the value in creating and maintaining a good reputation towards different stakeholders, including customers, partners, community, as well as existing and potential employees. By contributing to a specific community, an organization can improve the trust of customers and partners. The organization can demonstrate relevant technical knowledge about the OSS, and gain the influence needed in order to steer the development in a way that aligns with expectations and wishes from the customers and partners. Both retention and attraction of new customers and partners are further mentioned as potential results (CO1).

Increasing transparency is an aligning objective (CO2) in order to build trust among customers, or even the public. Being transparent about how the quality of a service is measured or what is produced based on public funding are two examples that are highlighted in interviews.

Table 4 Overview of the contribution objectives, related key benefits, in which case organization they were identified, and examples of similar findings in literature

ID	Contribution objective	Key benefits	CaseOrg.	Lit. ref.
Reputation-centric Objectives				
CO1	Prove skill and influence	• Improved trust towards customers.		
		• Improved trust towards partners.	1,2	Henkel (2006), Stuermer et al. (2009), Dahlander and Magnusson (2008), and Linåker et al. (2019a)
CO2	Increase transparency	• Improved trust among customers. (CaseOrg1)		
		• Improved trust among the public. (CaseOrg3)	1,3	Dahlander and Wallin (2006), Stuermer et al. (2009), and Linåker et al. (2019a)
CO3	Improve employer branding	• Attraction of talented employees.		
		• Retention of existing employees.	1,2,3	Henkel (2006), Stuermer et al. (2009), and Linåker et al. (2019a)
CO4	Be a good open source citizen	• Idealistic satisfaction among employees.	1,2,3	Henkel (2006), Jansen et al. (2012), and Linåker et al. (2019a)
Supplier-centric Objectives				
CO5	Create price pressure	• Lower subscription costs of procured products and services.		
		• Lower prices on tenders.	1,3	
CO6	Outsource infrastructure operation	• Internal focus on activities related to core business.	1	
Strategy-centric Objectives				
CO7	Collect data	• Improved machine learning and artificial intelligence algorithms.		
		• Creation of solutions based on Open Data sources.	1,3	
CO8	Standardize solution	• Force competitors to adapt and steer market or community according to internal agenda. (CaseOrg1-2)		
		• Improve competition and enable more value-adding development on market. (CaseOrg3)	1,2,3	Lindman et al. (2009), Jansen et al. (2012), Henkel 2006, West (2003), Linåker et al. (2018, 2019a)

Table 4 (continued)

CO9	Build a software ecosystem	• Enable and stimulate creation third party applications and services.	2,3	West (2007, 2008), Hartmann and Bosch (2016)
CO10	Improve partner collaboration	• Increased value of software ecosystem.	2	
Engineering Objectives				
CO11	Open up innovation process	• Accelerated innovation process. (CaseOrg1-3)		
		• Creation of more and better market-oriented solutions. (CaseOrg3)	1,2,3	Munir et al. (2016), Andersen-Gott et al. (2012), Linåker et al. (2018, 2019a)
CO12	Extend development resources	• Focus on more value-adding development.		
		• Accelerated development.		
		• Lower maintenance cost.	1,2,3	Munir et al. (2018a), Linåker et al. (2018), Stuermer et al. (2009), Henkel (2006), Lindman et al. (2009), and Olsson and Bosch (2017)

Allowing and enabling engineers to contribute to OSS can benefit the organization both in terms of retention and attraction of new talent (CO3). Reported examples include the ability to build a public CV, having an impact and collaborating with people outside the organization. Idealistic satisfaction of "doing the right thing" by contributing back and not just consuming OSS is also viewed as an important aspect among an organization's employees (CO4).

4.1.2 Supplier-centric Objectives

Supplier-centric objectives focus on how an OSS software artifact can be leveraged to better extract value from supplier-relations. For example, releasing projects as OSS can be a way to put price pressure on those providing corresponding products, as well as a means of inviting more potential suppliers to bid on tenders and thereby make the price more competitive (CO5). However, price pressure does not have to be the main incentive in terms of suppliers. By making a project available as OSS it becomes possible for cloud providers to run and maintain the project at scale and thereby lower the cost, while enabling internal resources to focus on more value-adding activities (CO6).

4.1.3 Strategy-centric Objectives

The strategy-centric objectives consider contributions that can have a larger impact on the organization's ability to stay competitive in the business environment where it operates (DaSilva and Trkman 2014). Releasing a software artifact as OSS may, for example,

Table 5 Overview of the contribution complexities, related key concerns, in which case organization they were identified, and examples of similar findings in literature.

ID	Contribution complexity	Key concerns	Case Org.	Lit. ref.
Control-centric Complexities				
CC1	Impact on value proposition	• Risk for misalignment between internal and external agendas on OSS with high impact on value proposition.		
			1,2	Linåker et al. (2018, 2019a
CC2	Impact on internal operations	• Risk for misalignment between internal and external agendas on OSS with high impact on internal operations.	1,2	Munir et al. (2018a), Linåker et al. (2019a, 2018)
IPR-centric Complexities				
CC3	Differentiating functionality	• Risk of loosing product-based competitive edge. (CaseOrg1-2)		
		• Risk of loosing process-based competitive edge. (CaseOrg1-2)		
		• Risk of damaging the business models of existing actors on market. (CaseOrg3)	1,2,3	Linåker et al. (2018), Wnuk et al. (2012), Van der Linden et al. (2009), Henkel (2006, Henkel)
CC4	Commoditization	• Risk for giving away competitive edge or differentiating functionality too early or an alternative solution being accepted before ones' own. (CaseOrg1-2)		
		• Risk of damaging the business models of existing actors on market. (CaseOrg3)	1,2,3	Van der Linden et al. (2009), Bosch (2013), and Linåker et al. (2018)
CC5	Sensitive IPRs	• Risk of giving away patented, patentable or in other ways sensitive IPR.	1,2	Linåker et al. (2018)
CC6	Substitutes	• Unnecessary cost of contributing if existing alternatives are considered as good or better.	1,2,3	Linåker et al. (2018)

Table 5   (continued)

CC7	License compliance	• Risk of violating license conditions if software artifact is not contributed.	1,2,3	Henkel (2006) and Linåker et al. (2018)
Exposure-centric Complexities				
CC8	Ethical use	• Risk of creating negative exposure, hurting brand and public trust.	3	
CC9	Security threats	• Risk of exposing security-related vulnerabilities.	1,2,3	(Linåker et al. 2018)
Cost-centric Complexities				
CC10	Budget and resource constraints	• Cost for preparing, contributing and maintaining software artifact.	1,2,3	Kilamo et al. (2012), West and O'Mahony (2005), and Dahlander and Magnusson (2008)
CC11	Modularity and architecture	• Technical feasibility to modularize and abstract software artifact.	1,2,3	(Linåker et al. 2019a)
CC12	Code aligment	• Cost of maintaining internal fork. Misalignment between internal and external development may prevent or complicate future contributions.	1,2	Munir et al. (2018a) and Linåker et al. (2018)
Community-centric Complexities				
CC13	External interest	• Risk of contribution not being accepted or community not being established.	1,2,3	Linåker et al. (2019a) and Linåker et al. (2019b)
CC14	Influence in community	• Low level of influence in the community may prevent or complicate the contribution of a software artifact.		
		• Target foundation may require a governance model too open which may render in too large scope and loss of control.	1,2,3	Munir et al. (2016), Schaarschmidt et al. (2015), Mäenpää et al. (2018), and Linåker et al. (2019b)
CC15	Community health	• Not contributing may have a negative impact on the health of the OSS project.	1,2,3	Linåker et al. (2019a), Jansen et al. (2012), and Zhou et al. (2016)

enable generation and collection of data that can be used to improve machine learning and artificial intelligence algorithms, while also enabling the potential development of new products or services based on the data (CO7).

A more control-focused objective (CO8) concerns the creation of a standard solution, either within an industry or within a specific community. If a solution gains traction and acceptance, it could provide a first-mover advantage as other actors would have to adapt. It may also provide the organization with some level of influence on the development of the artifact, with the potential to steer the direction of either a community or industry.

A related and potentially overlapping objective (CO9) may be to create a software ecosystem (Jansen et al. 2009) with the rationale of attracting and enabling third-party developers to create complementary products and services, and potentially also contribute to the platform constituted by the OSS project released by the platform provider. For example, releasing tools and infrastructure projects related to the platform, could further help to improve collaboration with partners and third-party developers, and thereby help increase the value of the software ecosystem (CO10).

4.1.4 Engineering-centric Objectives

The engineering-centric objectives concerns contributions where the rationale is to explicitly exploit the knowledge and resources of a community to one's advantage. One such objective (CO11) is to open up the innovation process by using the concerned community as a source of e.g., new and innovative requirements and feature implementations. An overlapping objective (CO12) is to use the development resources of the community as a force multiplier in order to share maintenance and quality assurance, while also accelerating development and enabling a value-adding focus for engineers internally of the organization.

4.2 Contribution Complexities

In total, 15 CCs were identified in the analysis process divided over five categories: control-centric complexities, IPR-centric complexities, exposure-centric complexities, cost-centric complexities, and community-centric complexities. Below, an overview is given to the CCs in each category. A more detailed overview is presented in Appendix D where the CCs are described separately with examples per identified concern, along with identified mitigation strategies for managing the complexities. Quotes from interviewees of the case organizations are given in Appendix D to provide context for each of the examples.

4.2.1 Control-centric Complexities

Control-centric complexities highlight the risk of the development of the software artifact (if released as OSS) heading in another direction than expected by the focal organization and what impact this would have. This is a risk that may be expected as the stakeholder population in a community is heterogeneous with agendas that not always align (Linåker et al. 2019a). Also if one wants to grow a community, the organization to some extent have to consider the wills and opinions of the community (Laurent and Cleland-Huang 2009). One complexity (CC1) considers the case when the artifact has a close relationship to the organization's value proposition, for example by enabling the organization to sell and deliver its products and services. Another complexity (CC2) considers the case where the artifact may have a more distant relationship to the value proposition but still being of importance in terms of internal operations. Examples highlight OSS projects integrated into the continuous integration tool-chain and used heavily by the product development departments. To address this complexity and the risk of deviating agendas in a community, and organisation can try to build influence in the concerned community through active engagement and contributions.

4.2.2 IPR-centric Complexities

Complexities centered around Intellectual Property Rights (IPR) considers what impact it would have on the focal organization's rights if the software artifact is released as OSS. One complexity (CC3) highlights the potential presence of differentiating functionality which could lead to a loss of competitive edge. A recommended practice is to separate between what is commodity and differentiating functionality, for example, through a modular architecture, and contribute underlying parts such as frameworks and libraries. Another complexity (CC4) considers the timing aspect of when the artifact should be released as OSS. It refers to the commoditization cycle where technology moves from an innovative and differentiating state to becoming commodity and general knowledge. One interviewee described it as a *"a trade-off between a competitive edge and burdon"* (I1) implying that by keeping it closed, an organization may guard what it considers differentiating, but as a consequence needs to maintain it by themselves, and at the same time risks that a competing solution gets released and adopted. It is, therefore, important to keep an awareness of potential alternative solutions.

From a public sector perspective, the intention may be the opposite in terms of CC3-4, for example to enable companies to create value based on public assets. This may also help concerned organizations to shift focus from commodity to more value-adding development. However, a risk is that it may damage the business and be viewed as the agency is intending to compete on the market. It may therefore be good to inform concerned organizations in advance and provide them with an opportunity to adapt and enter a dialogue when needed.

Another complexity (CC5) highlights sensitive IPRs in general, which may include patents used in a defensive patent portfolio, as a source of license revenue, or patents belonging to third-party. Hence, a critical review process for the need of the patents and their origin may be motivated in order to weigh these aspects against the potential value that the artifact may produce, if released as OSS.

A frequent scenario reported is that engineers request to release an artifact as OSS when there is already an acceptable alternative available as OSS (CC6). Actively encouraging engineers to research available options is therefore recommended. Education is also seen as a key approach to managing the risk of violating license conditions as these may demand that certain artifacts are contributed or made available as OSS under a certain license (CC7).

4.2.3 Exposure-centric Complexities

Exposure-centric complexities highlight ways in how the software artifact may be used that could have a negative exposure of the organization. The artifact may, for example, be used in contexts or for certain purposes that may be considered unethical and not originally intended by the organization (CC8).

Another risk is that security vulnerabilities are identified before they are fixed which could be used to damage the focal organization and other users of the artifact when available as OSS (CC9). It may therefore be good to pro-actively investigate potential unethical use cases and perform security audits, before releasing any artifact as OSS.

4.2.4 Cost-centric Complexities

Constraints in terms of budget and available resources can be an issue as preparing an artifact to be contributed, pushing it through a contribution process, as well as potentially

maintaining it once released as OSS has attached costs (CC11). In certain cases the technical feasibility of releasing the artifact may be questioned, as it may be too integrated into an organization's internal software stack, which in turn leads to that the cost may outweigh the potential benefits of releasing it as OSS (CC12). A mitigation can be to, early on, consider the potential of releasing the artifact as OSS and develop the artifact with that in mind, e.g., by using modular architecture and keeping code comments general.

When an artifact relates to an existing OSS project, there is an alternative cost of maintaining the artifact internally and a risk of growing a technical debt by not contributing (CC12). One way to minimize such costs and risks is to keep the internal fork as close as possible to the up-stream OSS project.

4.2.5 Community-centric Complexities

One risk is that the external interest is limited for a software artifact intended to be released as OSS (CC13). For existing communities, this would prevent it from being accepted. If creating a new community is considered, but none interested in joining, some of the expected benefits may be missed (e.g., shared maintenance costs). It may therefore be recommended to investigate whether there actually exists external interest, and if the use cases that the software artifact addresses are general enough.

Limited interest from a community may be due to misaligned agendas among its members (CC14). In these cases, it may be important for the focal organization to build up a level of influence through active engagement and contributions, in order to be able to contribute its artifacts.

When an organization is dependent on an OSS project and the community's health may be considered as a risk, it is important to contribute to that community (CC15). Hence, the level of health of the related community should be weighed against other complexities, when deciding if an artifact should be released as OSS.

5 Discussion

In this section we discuss the identified contribution considerations, in relation to literature, and with respect to the contexts they were observed.

5.1 Contribution Objectives

The expected benefits from sharing a software artifact as OSS is reflected by the identified COs and related key benefits (see Table 4). Objectives can be considered to cover both the idealistic, survival and commercial motivators highlighted by Jansen et al. (2012). Supportive findings can to a large extent also be found in literature if compared to Section 2.1. For example, the idealistically motivated objective CO4 - *Be a good open source citizen*, implying that an organization should respect and understand the needs, governance and culture of the community (Butler et al. 2018; Duc et al. 2017 Dahlander and Magnusson 2005, 2006; ÅGerfalk and Fitzgerald 2008). Further, the commercially motivated cost-saving objectives implied by the extended development workforce (CO12), and related benefits, has also been observed in a number of other studies (e.g., Munir et al. 2018a; Stuermer et al. 2009; Henkel 2006; Lindman et al. 2009; Olsson and Bosch 2017; Van der Linden et al. 2009). Support was also found for the survival, or more strategically, motivated objectives such as

CO9 - *Build a software ecosystem* (West and Gallagher 2006, 2008). Some objectives, however, were not reflected among the reviewed literature, e.g., CO5 - *Create price pressure* on third-party vendors and suppliers.

A challenge with the intangible (Shahrivar et al. 2018), or non-monetary benefits (Morgan and Finnegan 2014) is that they may be less obvious and harder to measure and track in financial spreadsheets (Morgan and Finnegan 2014), compared to the tangible revenues and monetary benefits. In this study, an attempt is made by presenting key benefits to each objective. Future research could strive to identify and derive suitable and actionable metrics for the different types of benefits, or objectives[8]. With such metrics in place, contribution requests and strategies can potentially become easier to motivate, execute, follow-up and learn from. Morgan and Finnegan (2014) highlight how organizations *"...need to put structures and incentives in place to convert intangible asset capture into something tangible for the [organization]"*.

Objectives further align with the propositions proposed by Munir et al. (2018b) which states that organizations adopting OSS in their tools and infrastructure setups may benefit through reduced product development costs and time-to-market, as well as increased product and process innovation. Considering Munir et al. (2018b) classification of why and when organizations adopt and share software as OSS, all three case organizations can be classified as having a proactive approach with the main focus on building symbiotic relationships through their OSS engagements. Similar to Sony Mobile (Munir et al. 2018b), these organizations can hence be seen as mature players where the use and sharing of OSS are motivated by business rationale.

When comparing the two private case organizations, CaseOrg1 and 2, against the public case organization CaseOrg3, many of the objectives overlapped, although the expected key benefits may differ. For example, regarding CO8, which is focused on creating or replacing an existing industry or community standard, CaseOrg 1 and 2 see value in being able to steer the direction of a community or industry, and potentially disrupting competitors by commoditizing a certain technology (Van der Linden et al. 2009). CaseOrg3, on the other hand, views the key benefit as improving competition in the industry or market and potentially forcing the private actors to adapt. Hence, both private and public actors may view the release of OSS and commoditization of the underlying technology as a strategic tool, but in some cases for the opposite reasons.

5.2 Contribution Complexities

The contribution complexities presented in Table 5 can be viewed from the different perspectives of a contribution strategy. With respect to whether a software artifact or parts of it should be shared as OSS or not, a common concern in literature regards the risk of sharing differentiating or in other ways sensitive IPR ((e.g., (Wnuk et al. 2012; Van der Linden et al. 2009; Henkel 2006; 2008; West and Gallagher 2006; Iivari et al. 2008)). This risk was also explicitly recognized in the complexities CC3 and CC5. Selective revealing, as labeled by Henkel (2006), was a recognized practice in both CaseOrg1 and 2, as well as in Sony Mobile (Linåker et al. 2018).

In contrast, this was not an issue for CaseOrg3. As a public sector organization, they aim to release what they find most "differentiating" in their view, and what can provide the biggest value to their ecosystem, and in the end, the citizens. Their motivation lies in their

---

[8]See e.g., https://chaoss.community/ Accessed: July 8, 2020

goals as defined by the government: *"to facilitate and enable matching between job-seekers and employers on the Swedish labor market"*[9]. However, they do see a balance between raising the bar for private actors in their ecosystem and not hurting their business model. Although not explicitly found in the coding, this introduces a timing factor, i.e., that the contribution or release of the software artifact is stalled until the private actors have had time to adapt. This addresses the question of when a software artifact should be shared as OSS.

A related complexity is that of the commoditization process and the concerned software artifact (CC4), which is also brought up in literature (Van der Linden et al. 2009; Wnuk et al. 2012; Haruvy et al. 2008; Kort and Zaccour 2011; Caulkins et al. 2013; Linåker et al. 2018). By being early, an organization can get a first-mover advantage, gain bigger influence, and potentially steer the development, e.g., within a community through a new feature (Linåker et al. 2019a; Munir et al. 2018a), or within an industry through a new standard or platform (West and Gallagher 2006) (see e.g., contribution objective CO8 and CO9). By being too late, an organization can risk having to adapt to competing solutions or maintain the internal solution (Linåker et al. 2018; Ven and Mannaert 2008). Among the case organizations, both CaseOrg1 and 2 experienced this complexity. I1 from CaseOrg1 described it as a *"...trade-off between competitive edge and burdon"*, aligning with other reported observations (Wnuk et al. 2012).

Lastly, the question of where, i.e., if the software artifact should be contributed to an existing community, or if a new community should be established, touches on several of the complexities, many of which are found across all three case organizations. A common concern among many of the complexities is the need for, or risk of losing, control (e.g., CC1-2, 14). In these cases, if an organization requires a certain level of control on the development direction of the software artifact, the artifact may preferably be released in a community where the organization has a high level of influence on the RE process (Linåker et al. 2019a; Schaarschmidt et al. 2015), or as a new community (Linåker et al. 2018). A decisive factor is the external interest for the software artifact (CC13), i.e., whether or not it be accepted within a specific community, or if there is an interest for a new community. In the former case of an existing community, having an existing influence within the community could help to create the interest (CC14) (Munir et al. 2018a). The health of the OSS community is another complexity as a community requires active contributions from its members to stay alive. These factors may be considered in a specific *community strategy* as seen from the perspective of the focal firm, which outlines the importance of that community, and what engagement and level of influence that the focal firm wants with respect to the community's requirements engineering process (Linåker et al. 2019a).

When weighing the options of contributing to an existing community or creating a new one, the earlier may be preferred if possible. As highlighted by CC10, creating a new community may be related to a higher cost, and comes with a number of extra challenges (Kilamo et al. 2012; West and O'Mahony 2005). How this is best done is, however, beyond the scope of this paper, and an interesting topic of further research.

6 Threats to Validity

The identified contribution objectives and complexities are the result of a multiple-case study of three case organization over the lapse of two research cycles. To determine the

---

[9]https://arbetsformedlingen.se/om-ossAccessed: July 8, 2020

external validity (Runeson et al. 2012) of the objectives and complexities, the characteristics of these organizations need to be considered as they define the problem context (Wieringa 2014) on which the COs and CCs are based.

The three case organizations investigated in this study both have overlapping and distinct characteristics. To some extent, they provide extremes (Flyvbjerg 2007) to each other, while still having resembling characteristics. Considering the way they use and leverage OSS, all three organizations use it for their internal tool and infrastructure setups. In CaseOrg2, the department developing these tools are explicitly studied. CaseOrg1 uses OSS to enable and add value to their hardware devices and as a basis for certain services sold to business-oriented customers. As a public agency, CaseOrg3 differs from CaseOrg1 and CaseOrg2 in that they are not driven by commercial business incentives. Instead, they wish to help private actors focus on more value-adding activities and thereby improve job-matching capabilities for employers and job-seekers. CaseOrg2 however, does not consider themselves as having any product differentiation, compared to CaseOrg1. Following the categorization by Munir et al. (2018b), the organizations provide a suitable sample as they all have an awareness for why they share software as OSS and may reflect on the complexities that are present, even in the case of CaseOrg3 who does not have an explicit contribution process in place.

We acknowledge the limitations of case studies and do not claim any statistical generalization (Runeson et al. 2012). However, we do believe they provide a means to gather deep knowledge of industry practice and rationale in the problem context. By considering the case organizations' characteristics, the reader can put the presented contribution objectives and complexities as well as the organizations' rationale and concerns for sharing software as OSS into context. Through analytical generalization (cf. analogical inference (Wieringa 2014)), results from this study can then be extended to cases with similar characteristics within a similar context (Runeson et al. 2012). Both similarities and dissimilarities between the source and target cases should be thoroughly analyzed (Ghaisas et al. 2013). In Section 3.1, we provide a general description of each of the case organizations, as well as of their specific contribution processes and examples of OSS projects that they are contributing to or have released as OSS. Quotes from interviewees (see examples in Appendices C and D) are used to describe the contribution objectives and complexities, and to provide further contextual factors that may otherwise risk being lost in the reporting of the research if abstracted by the researcher.

A threat in regards to reliability relates to that only the first author was involved in the data gathering and analysis process of the study. To minimize the risk of misinterpretations, member-checking (Easterbrook et al. 2008) was performed by presenting interview summaries to key interviewees at the case organizations. During the coding process, the inductive approach infused a constant comparison between new data and existing codes, enabled by audit-trails being kept (Runeson et al. 2012). Cross-case analysis (Seaman 1999) was also performed as codes identified in one case were reiterated in the transcripts from the other cases, after which a final coding scheme assembled. As can be noticed in Table 3, there was saturation in the number of emerging contribution objectives and complexities. Triangulation (Runeson et al. 2012) with archival data was also performed by cross-checking coding schema and basing interview questionnaires on contribution request forms from seven organizations.

After the final coding was performed in the second research cycle, the contribution objectives and complexities were presented and discussed with I1 from CaseOrg1, I7 and I8 from CaseOrg2 and I15 from CaseOrg3. This kind of static (Gorschek et al. 2006) or descriptive validation (Hevner et al. 2004) is a useful approach in the artifact validation phase to gain

feedback and refine a research artifact before it is transferred or implemented in a real-world problem context (Wieringa 2014; Gorschek et al. 2006).

I1 from CaseOrg1 confirmed identified contribution objectives and complexities. I1 added further facets to e.g. CC6, that patents may also provide a source of license revenues in other cases which should also be considered. I1 found the objectives and complexities useful and that *"[they] really fit into [CaseOrg1's] strategy which is, we are trying to streamline the entire [contribution] process so that we are asking the right questions and we want to get to a point where we are able to approve everything online and don't need to have a meeting. Because if these questions are answered correctly then we should be able to even have an AI/ML-based algorithm which says, the risk is 10% so it is OK to approve it"*. I1 also adds that a contribution may be *"more things than just code"*, exemplifying evangelizing, technical writing, writing bug reports, sharing of knowledge and experience, as well as test cases and design documentation. We agree with I1 that a contribution may be many things, but choose to limit the scope to the sharing of software artifacts as OSS. We view these artifacts as internally developed software functionality of different size and complexity, e.g., bug-fixes and features to frameworks, projects, and products, including e.g., related test cases and documentation. Other activities we believe should be covered by a community strategy that specifies how an organization should engage with a specific OSS community (Linåker et al. 2019a).

I7 and I8 from CaseOrg2 found the contribution objectives and complexities valuable and saw value in comparing with other organizations. They believed that the objectives and complexities can be used as an eye-opener to those within the organization that are skeptic to OSS. Both interviewees agreed with and verified the relevance of all of the identified objectives and complexities. They also added support for CC5 and CC15 and refined the description of CO7.

I15 from CaseOrg3 confirmed the identified objectives and complexities, while also adding support for CC4 and CC5. In regards to the latter complexity, I15 highlights how it also connects to CC4 and how they strive to share software artifacts that are differentiating in order to help businesses focus on more value-adding activities.

In conclusion, we believe that the identified contribution considerations are relevant to the studied case organizations, but we also believe that the findings have relevance beyond these organisation, while that of course depends on problem context to which these findings are transferred. The validation of the completeness of the set of contribution considerations, is a matter for continued research, but our investigation of existing literature has not indicated any significant omissions.

7 Conclusions

In this study, we aim to help organizations in deciding if a software artifact (e.g., feature, framework or project) should be released as OSS, and, if so, when and where. For a specific artifact, the "*what*"-question regards if the artifact should be contributed in full or kept closed, or if certain parts can be contributed under certain conditions. The "*when*"-question regards the timing of the release. Finally, the "*where*"-question regards whether the artifact should be contributed to one of many existing OSS communities or if a new community should be established. Answers to these questions may be valuable input to the development of a *contribution strategy* for the concerned software artifact.

To support organizations in creating effective contribution strategies, we conducted a multiple-case study at three software-intensive organizations using an iterative approach spanning over two research cycles. A set of 27 contribution considerations, divided into 12 objectives and 15 complexities, were identified based on an inductive and iterative coding of 20 interviews across the three case organizations.

Contribution objectives highlight opportunities for 1) improving reputation towards community, customers, partners, and current and future employees, 2) managing suppliers through price pressure and outsourcing, 3) managing partners and competitors through standardization efforts and ecosystems, and 4) exploiting externally available knowledge and resources through open innovation and extended development resources.

Contribution complexities focus on 1) risk of losing control of the development of business-critical software in concerned communities, 2) risk of giving away differentiating IPR that provides a competitive advantage, 3) risk of unintentionally exposing unethical use-cases and security vulnerabilities, 4) costs of abstracting and generalizing software artifact, pushing through contribution process and potentially maintaining once contributed, and 5) difficulties in deciding where the artifact should be contributed based on external interest, potential need for influence, and community health.

Future work should look to generalize these identified contribution considerations beyond the problem context of the three investigated case organizations. In order to arrive at a framework that can be used by software-intensive organizations when setting up their contribution strategy decision process, further validation and generalization is need. Specific research focus is also needed to investigate possible relationships between contribution objectives and contribution complexities. In addition, research attention could be given to identifying and formalizing metrics that may be used for quantifying the potential benefits proposed in the objectives, as well as the risks and costs implied by the complexities. Such metrics could help organizations arrive at key performance indicators, such as a Return-on-Contribution, which may help to make decisions comparable, measurable, and easier to motivate. This could further help organizations in automating their contribution processes and thereby potentially lowering the threshold for developers to contribute, and also, hopefully, help the organization to more effectively reach their contribution objectives.

Appendix A: Questionnaire - First Research Cycle

Demographics:

- What is your job title?
- How many years of experience do you have in your current and similar roles?
- Could you, in short, describe your daily work and responsibilities?
- Could you, in short, describe your experience in working with OSS communities?

Contribution Strategy:

- Do you, in any way, consider what you contribute and reveal as open source? If yes, how? Is it formalized in any way? How could this be improved/otherwise done? What connections do you see to a company's ROI and competitive edge?
- How would commoditization affect what you contribute and not? How would it affect the timing of when you contribute?
- How would a feature's profit for the company affect what is contributed and not? How would it affect the timing of when you contribute? How could this aspect be judged or quantified?
- How would it affect if the feature or the knowledge and technology behind it is hard to acquire or control? How could this aspect be judged or quantified?
- How would you reason in regard to if and when to contribute, given that a feature results in a:

    - high profit for your company, and is critical to maintain control over?
    - low profit for your company, and is critical to maintain control over?
    - high profit for your company, and not critical to maintain control over?
    - low profit for your company, and is not critical to maintain control over?

- What other aspects would affect your decisions?
- How would your decisions affect how you engage and invest in the community where the feature is to be contributed to?
- How are policies or decisions regarding what to contribute communicated today? How could this be improved/otherwise done?
- Is there anything I have missed that you would like to pick up on? Or anything else that you would like to talk about?

B Questionnaire - Second Research Cycle

Demographics:

- What is your job title?
- How many years of experience do you have in your current and similar roles?
- Could you, in short, describe your daily work and responsibilities?
- Could you, in short, describe your experience in working with OSS communities?

Contribution Objectives:

- What reasons do you see to share your internally developed software as OSS? What can your organization gain out of this? How can you measure it?
- What reasons do you see in relation to competition and collaboration with external parties? Who would it benefit or hurt?
- What effect can you have on a third party by releasing software as OSS? For example, competitors or suppliers of similar proprietary products?

– Do you consider if there is a potential to create a new standard? How can such potential be measured?
– Are there any type of software that should be shared for different reasons? How would these types be characterized?

Contribution Complexities:

– What aspects should you take into consideration in the decision of sharing software as open source or not?
– What consideration should be taken to how software relates to your value proposition and how it affects your revenues? For example, if the software is shipped with your product or if it is used to build or deliver your product/service?
– How is a decision affected if the software contains differentiating parts?
– How is a decision affected if the software contains patents or parts that could be patented?
– Is there any type of sensitive information within the software that could complicate a contribution?
– How should any competitive advantages be taken into consideration?
– How do timing and commoditization affect such aspects and any future decisions? How do you consider the risk of competing solutions being released before yours?
– How do you consider internal restrictions in terms of budget and resources? What costs do you see related to releasing software as OSS? How do these costs affect the decision?
– Do you see any other business related aspects, impediments or risks that should be taken into consideration?
– What consideration should be taken in regards to how the software is used and integrated into your organization and your product/services?
– What consideration should be taken to security-related aspects?
– How do you consider the software's generalizability and potential to extend?
– Do you see any other technical aspects, impediments or risks that should be taken into consideration?
– How is a decision affected in terms of the need to control the development of the software?
– How do you consider existing alternatives (OSS and proprietary)?
– What alternatives to you see if there is a shallow external interest from a community?
– What factors affect your decision when choosing to contribute to an existing community or creating a new community?
– How do you consider the health and sustainability of a community?

C Findings on Contribution Objectives

C.1 Reputation-centric Objectives

C.1.1 CO1 - Prove skill and influence:

Description: Being an active contributor in a community can help to build a reputation of an organization as technically skilled and influential in the community (Munir et al. 2016; Ven and Mannaert 2008).
Benefit: Improved trust towards customers.

Example [CaseOrg1]:   *"We basically realized that we were so dependent on [OSS project] and that it was the selling point of our product, so we needed to demonstrate for customers that we were one of the core contributors. It was a selling point in the marketing brochure"* (I1).

Benefit: Improved trust towards partners.

Example [CaseOrg2]:   I7 reports how being an active contributor in a community can help to promote CaseOrg2 in new channels and build up new partnerships.

C.1.2 CO2 - Increase transparency:

Description: By being transparent in how an organization performs certain actions, they can build trust among customers and potentially avoid allegations of wrongful doing.

Benefit: Improved trust among customers.

Example [CaseOrg1]:   A tool was developed and open sourced to measure the speed of their services. By keeping the project open, the organization could be transparent in how they performed the measurement.

Benefit: Improved trust among the public.

Example [CaseOrg3]:   From a public sector perspective, CaseOrg3's aim is in a similar manner to create transparency towards the public and thereby earn their trust, *"this is what you get for your tax money"* as put by I13.

C.1.3 CO3 - Improve employer branding:

Description: Working with OSS projects can help an organization to both retain existing and attract new engineers (Munir et al. 2016).

Benefit: Attraction of talented employees.

Example [CaseOrg1]:   *"It is important to people and I think a positive employment characterization that you get to engage with open source communities and that the company does release something as open source projects. I think that is a big selling point that people are looking for in whom they want to work for as an engineer"*.

Benefit: Retention of existing employees.

Example [CaseOrg1&3]:   Besides being allowed to engage with an external community with the intrinsic rewards that follow, highlighted by I19 as *"the fun parts"*, working with OSS offers engineers the opportunity build their own transparent portfolios. As stated by I6, *"[the engineers] view it as a core part of their career development and in a way, they can kind of be recognized for that in a very different way than they could if they were simply working on internal or commercial software"*.

C.1.4 CO4 - Be a good open source citizen:

Description: Contributing to an OSS project may for some be ideologically motivated (Jansen et al. 2012).

Benefit: Idealistic satisfaction among employees.

Example [CaseOrg1]:   *"I think it is from one point of view the right thing to do, because you are taking advantage of this set of code, and so you should, while you are taking advantage of it, you should also be contributing back [from] a good citizen point of view"*. I5 further adds that a reason *"may be that the developers are pro-OSS and so they are just like, this isn't secret sauce, so just sort philosophically we want to contribute this*

to the commons because maybe someone will take advantage of it. There's a good citizen aspect to it".

C.2 Supplier-centric Objectives

C.2.1 CO5 - Create price pressure:

Description: Releasing a project as OSS can be a way to put price pressure on third-party vendors and suppliers of proprietary solutions.

Benefit: Lower subscription costs of produced products and services.

Example [CaseOrg1]: As a reaction to an expensive proprietary infrastructure solution, CaseOrg1 developed an internal replacement and open sourced it as *"a competitive advantage in the negotiations against [the supplier of the proprietary solution]"* (I4) in order to *"drive the cost of doing business down"* (I5).

Benefit: Lower prices on tenders.

Example [CaseOrg3]: From a public sector perspective, by releasing software as OSS, and requiring tenders to build on it, competition in the tender process is improved as smaller actors enabled to participate in a process *"otherwise set-up to benefit large firms"* (I12).

C.2.2 CO6 - Outsource infrastructure operation:

Description: Sharing software as OSS can be seen as a way to not just outsource the development of a software project (ÅGerfalk and Fitzgerald 2008), but also the operation of it.

Benefit: Internal focus on activities related to core business.

Example [CaseOrg1]: As explained by I5, *"We can get to a point where we can actually outsource the operations of this [software] ... via an open source path, which then frees up this team to do other stuff"*.

C.3 Strategy-centric Objectives

C.3.1 CO7 - Collect data:

Description: By sharing e.g., algorithms as OSS, others can generate data which can then be used to improve artificial intelligence and machine learning initiatives.

Benefit: Improved machine learning and artificial intelligence algorithms.

Example [CaseOrg1]: I1 explains *"The Machine learning team has often come to us saying, I want to open source this algorithm because I wanna learn how it grows, and we alone don't have enough data to feed and we need the world to feed it data"*.

Benefit: Creation of solutions based on Open Data sources.

Example [CaseOrg3]: I15 exemplifies how the gathering of job-ads can help create an automated review-function which could classify if an add is discriminatory or not.

C.3.2 CO8 - Standardize a solution:

Description: By having an OSS project or contribution adopted and accepted as a standard solution, an organization can potentially replace existing proprietary and community solutions (Lindman et al. 2009; Jansen et al. 2012; Henkel 2006; West 2003).

Benefit: Force competitors to adapt and steer market or community according to internal agenda.

Example [CaseOrg1]: A project was developed and open sourced in order to replace proprietary solution as *"a way to get away from a commercial vendor and open up the access to the data collected by the vendor, but also to be able to influence and control [the project's] direction"* (I6). As further explained by I1, *"by putting it out there you can influence it and thereby where the market is heading"*, potentially steering the competition in the direction most beneficial to the firm.

Benefit: Improve competition and enable more value-adding development on market.

Example [CaseOrg3]: From a public sector perspective, CaseOrg3 views the benefit as *"improving competition"* (I13) and *"allowing more to join the market"* (I15). I15 continues, *"I think we are raising the bar for the entire market, allowing everyone to level up and stop focus on base infrastructure"*.

C.3.3 CO9 - Build a software ecosystem:

Description: Software released as OSS can serve as a technological platform around which a software ecosystem may form with firms who can start to collaborate and interact through a shared market of software and services.

Benefit: Enable and stimulate creation third party applications and services.

Example [CaseOrg]: From a public sector perspective, CaseOrg3 is aiming to create a software ecosystem by contributing internal software artifacts as new OSS projects along with Open Data sources. Based on the platform, constituted by the OSS projects and Open Data sources, private organizations and firms on the labor market can create new and *"hopefully better market-adapted solutions"* (I13). I17 adds *"I think there is an interest and a need for everybody within the sector to collaborate and to help each other and to make sure that the right individuals land the right opportunities. So what we are trying to gain is literally a collaboration [...] with others to provide better services to society"*.

C.3.4 CO10 - Improve partner collaboration:

Description: Sharing e.g., tools and infrastructure projects as OSS could potentially act as a complement and improve collaboration between CaseOrg2 and other actors within CaseOrg2's software ecosystem that is connected to its core-business of embedded devices.

Benefit: Increased value of software ecosystem.

Example [CaseOrg2]: As put by I7, *"It would be easier to collaborate around the same stack... Also, by offering boilerplates to our partners we help them to do the right thing from start"*. I8 adds, *"I think, would [CaseOrg2] be more open with its tools and libraries, this would be appreciated by, and a way to come closer to our third-party developers"*.

C.4 Engineering-centric Objectives

C.4.1 CO11 - Open up innovation process:

Description: Collaboration with OSS communities can be seen as a case of Open Innovation (Munir et al. 2016), allowing an organization to extend its R&D and innovation capabilities and initiate new collaborations in an open and transparent way.

Benefit: Accelerated innovation process.

Example [CaseOrg1]: I6 describes it as *"Open source is in many cases about doing external R&D in a way. You are leveraging external groups to do things, and it is often the case that many companies don't really fund medium to longer term R&D in-house anymore... You get the network effect of so many more participants"*.

Benefit: Creation of more and better market-oriented solutions

Example [CaseOrg3]: Open collaboration may also be seen as an opportunity to *"open up the requirement engineering process"* (I15) and create *"faster feedback-loops, and to become more reactive and compatible to the needs of the market"* (I12).

C.4.2 CO12 - Extend development resources:

Description: Using and combining the development resources of an OSS community with an organization's internal, software development may potentially be accelerated and extended as e.g., features are implemented and bugs corrected (Munir et al. 2018a).

Benefit: Focus on more value-adding development.

Example [CaseOrg1]: I6 describes it as *"the more that you can push things down to commodity, the easier it becomes, and cheaper, and then you can keep focusing more up the stack. Keep focusing on the next new feature"*.

Benefit: Accelerated development.

Example [CaseOrg1]: *"[Y]ou almost get this acceleration of innovation on that platform. It supercharges [the platform] in a way and you are getting, maybe it is us and five other actors, maybe even competitors, and we are all contributing back to things that we are finding. It is a better product for everybody. So I think that accelerates the development process"* (I6).

Benefit: Lower maintenance cost.

Example [CaseOrg1]: ''*The code is actually a cost and when you realize that, you will see that it is better to share that cost than keeping it to yourself"* (I12)

D Findings on Contribution Complexities

D.1 Control-centric Complexities

D.1.1 CC1 - Impact on the value proposition:

Description: Software artifacts that have a close connection to an organization's value proposition and its revenue stream may warrant special attention in order to determine any potential costs or negative risks that may be implied by contributing the artifact.

Concern [CaseOrg1&2]: Risk for misalignment between internal and external agendas on OSS with high impact on value proposition.

Example [CaseOrg2]: In the case of CaseOrg2, the department studied in this paper focuses on infrastructure and tools-projects which is used by CaseOrgs2's product development teams. The software developed and maintained by the department can hence be considered to have an indirect connection to CaseOrg2's value proposition and revenue streams, or as put by I7, *"not even close to core business"*. This provides the department with a *"much less restrictive process than what applies for core business"* (I7).

Example [CaseOrg1]: CaseOrg1 develops and maintains similar types of projects, but also those that have a stronger connection to the organization's value proposition and revenue streams. For example, one project that they have released is a Linux-based operating

system embedded in hardware devices which enables the organization the deliver its service offerings to its customers. Another project released by the organization makes up a pivotal part of their infrastructure, also enabling the delivery of its service offerings, but also the possibility to offer the infrastructure as a service to business customers, including competitors. In both examples, the projects are considered as commodity but of strategic importance why the organization needs to be *"actively involved and be able to steer the direction and make sure that as that project evolves that it continues to meet our needs and to evolve in a certain way"* (I5).

Mitigation strategy:　Build influence on OSS community's RE process to enforce internal agenda.

CC2 - Impact on internal operations:

Description:　Even though a software artifact may have a loose connection to a organization's value proposition and revenue stream, it may still be of strategic importance internally. By releasing the artifact as OSS there is a risk of control being too diluted due to other stakeholders' interests.

Concern [CaseOrg1&2]:　Risk for misalignment between internal and external agendas on OSS with high impact on internal operations.

Example [CaseOrg2]:　*"We are extremely dependent on [OSS project] as we have built our whole continuous integration tool-chain on it, same goes for [OSS project]. Hence, we need to be active so that we can affect the direction on the projects, otherwise, it could become extremely costly for us. Better tools enable us to make better products"* (I8).

Example [CaseOrg1]:　I1 phrases it as, *"Which projects are we actively using as a company that we are so dependent upon for our success that we need to be at the table? You can do a survey on the company and ask, what open source infrastructure are you using, and how critical is it to your success? And how are you engaging today? And what is missing in your engagement?"*. I4 adds, *"We do need to influence [the OSS project's] roadmap because as the project continues to evolve, and we are such a huge user of it, we still need to drive its roadmap quite a bit to be able to get maximum value from it"*.

Mitigation strategy:　Build influence on OSS community's RE process to enforce internal agenda.

D.2 IPR-centric Complexities

D.2.1 CC3 - Differentiating functionality:

Description:　Giving away software artifacts that provide product differentiation or other types of competitive edge is a common fear among firms, but also an opportunity and a balancing act for public organizations.

Concern [CaseOrg1&2]:　Risk of loosing product-based competitive edge.

Example [CaseOrg1]:　I3 explains, *"when I look at what we've approved so far, practically 93 percent of what's coming in front of the committee we approve. The 7 percent has been the things that are core to our business, or something that is a competitive advantage"*. This may be a consequence of CaseOrg1's work to actively encourage its developers to separate between differentiating and commodity functionality as explained by I6, *"These things are really basic functionality, that is ok, we are going to share that, that is part of the core platform. But there are maybe some parts where you can on a modular basis extend it or do integrations with internal systems in order to not give away the secret sauce, the differentiation"*.

Mitigation strategy: Identify and separate between differentiating and commodity functionality when possible.

Concern [CaseOrg1&2]: Risk of loosing process-based competitive edge.

Example [CaseOrg2]: For the department studied in CaseOrg2, a bigger concern relates to the risk of giving away process-based competitive edge, as explained by I7 *"Of course we have tools that can provide us with a competitive edge"*. Faster development pace and better quality assurance are two areas highlighted by I9.

Mitigation strategy: Identify and separate between differentiating and commodity functionality when possible.

Concern [CaseOrg3]: Risk of damaging the business models of existing actors on market.

Example [CaseOrg3]: Concerning CaseOrg3, they prioritize releasing software artifacts that have the potentially highest differentiating value for actors on the market. However, as expressed by I15, *"It is a balancing act. There will be cases where we will open up stuff that some make a living on. Actors will have to innovate their business models and adapt. Our aim is to improve competition, not hurt it"*.

Mitigation strategy: Inform actors and provide possibility to adapt in advance.

D.2.2 CC4 - Commoditization:

Description: The timing of when to release something as OSS can be a complex issue. As I1 highlights, *"There is a trade-off between a competitive edge and burdon"*. I.e., while keeping the software closed may provide a competitive advantage, it may cost in terms of maintenance and support.

Concern [CaseOrg1&2]: Risk for giving away competitive edge or differentiating functionality too early or an alternative solution being accepted before ones' own.

Example [CaseOrg1]: I1 describes as a *"waiting game... [Some] may want to say - I want to hold on to this feature longer - because they don't want to make it a level playing field for everybody, or they may take certain features and say, - I think we need to get it out there as fast as possible, because we heard that competitor X is working on putting this into the pipeline... So I think it could be both ways, there could be some features where we say, No, let us hold on to this until we get a first-mover advantage on the market using it, then once we have a significant advantage, then it is ok to commoditize it and put it out there"*.

Mitigation strategy: Maintain awareness in community and industry about potential alternative solutions.

Concern [CaseOrg3]: Risk of damaging the business models of existing actors on market.

Example [CaseOrg3]: CaseOrg3 also considers the commoditization process as an important aspect, but as highlighted in CC4 they strive to share software artifacts that are differentiating in order to help businesses focus on more value-adding activities.

Mitigation strategy: Inform actors and provide possibility to adapt in advance.

D.2.3 CC5 - Sensitive IPRs:

Description: Sensitive IPs or patents is not limited to functionality that provides a competitive edge in terms of a product or process (see CO4). They can also constitute pieces in a defensive patent portfolio, serve as a revenue source of license subscriptions, or belong to a third-party.

Concern [CaseOrg1&2]: Risk of giving away patented, patentable or in other ways sensitive IPR.

Example [CaseOrg1]:   For CaseOrg1, patents can be viewed as a competitive edge even in cases where they do not cover functionality that is actively used. As I1 explains, *"We live in a very litigious environment, which means, as a company we are in an industry where there are lots of lawsuits against each other, so we use our patent portfolio in a defensive way. So there is a drive to build a patent portfolio which is why the patent office encourages people to seek patents, because the bigger and better the patent portfolio, the more we can defend ourselves"*. Hence, a common question that is asked in CaseOrg1 is *"can and should we patent this?"* (I1). I1 also adds that in other organizations certain patents may provide license revenue, which should also be weighed against the potential value of sharing the patent as OSS.

Example [CaseOrg1]:   In the case of CaseOrg2, as they are orchestrating a larger software ecosystem around its products, a question they ask is if there are any patents or IPRs that belong to one of their partners.

Mitigation strategy: Critically review need and origin of patents.

D.2.4 CC6 - Substitutes:

Description: If there are existing alternatives to the software artifact available, it may be questioned why the artifact should even be considered.

Concern [CaseOrg1,2&3]:   Unnecessary cost of contributing if existing alternatives are considered as good or better.

Example [CaseOrg1]:   according to I3, *"One of the questions we ask in the open source contribution form is, is there an existing project that does the same thing or is this one something unique and new? We've had people that, say, I wrote an HTTP client, and I want to open source it, that question goes to that, hey, there are plenty HTTP clients, why are you writing another one?"*. However, sometimes it may be motivated if there is a strategic intent to replace an existing solution in a community or an industry standard.

Mitigation strategy: Educate developers to research substitutes and motivate why artifact still should be released as OSS.

D.2.5 CC7 - License compliance:

Description: In cases where the software extends or integrates with an OSS project, the OSS license of that project needs to be reviewed and respected. Copyleft and restrictive licenses may require that the artifact is contributed.

Concern [CaseOrg1,2&3]:   Risk of violating license conditions if software artifact is not contributed.

Example [CaseOrg2]:   I7 highlights, *"we must be compliant with the licenses we have in our source code"*.

Mitigation strategy: Implement compliance programs, educate engineers and automated license-checking.

D.3 Exposure-centric Complexities

D.3.1 CC8 - Ethical use:

Description: By releasing a software artifact as OSS, anyone is allowed to use it under the same conditions provided by the OSS license. Hence, a risk is that the software may be used for purposes not originally intended.

Concern [CaseOrg3]:   Risk of creating negative exposure, hurting brand and public trust.

Example [CaseOrg3]:   I18 sees a a risk of *"cases where we would not be very proud of as a public agency"* and asks *"how far does the responsibility of CaseOrg3 as a public agency stretch?"*

Mitigation strategy: Investigate potential alternative use cases of software artifact.

D.3.2 CC9 - Security threats:

Description: Releasing a software artifact as OSS may pose a security threat by exposing unknown vulnerabilities present in its source code.

Concern [CaseOrg1,2&3]:   Risk of exposing security-related vulnerabilities.

Example [CaseOrg2]:   At CaseOrg2, *"the security department has a positive view on open source as you get more eyes on the code"* (I11). However, they still have a careful review process in place as they do not wish to expose any potential back-doors that could lead to the organization's hardware-based products.

Example [CaseOrg3]:   At CaseOrg3, I18 highlights that *"it is the data that is considered valuable and needs protection"* which is rather done by *"proper key management"* than keeping any related software closed.

Mitigation strategy: Include security audits in contribution process.

D.4 Cost-centric Complexities

D.4.1 CC10 - Budget and resource constraints:

Description: Contributing a software artifact always comes with a cost, both in terms of preparing, contributing and potentially maintaining the software artifact as OSS.

Concern [CaseOrg1,2&3]:   Cost for preparing, contributing and maintaining software artifact.

Example [CaseOrg2]:   Abstracting, modularizing and generalizing an artifact may prove an expensive effort compared to projects that are developed with the intention from the start. As put by I8, *"if we build something that is tailored to and dependent on internal infrastructure, it is most often no idea to contribute it. In most cases, we can modularize and generalize it, but the cost can get really high, so it is a matter of if it is worth it or not"*. I9 further mentions that there *"is a lot of hidden costs"* that are connected to the contribution process, such as *"scrubbing"* or cleaning the source code and its version history of sensitive or unnecessary comments and information. Other costs include going through the contribution process of the related community, or creating a community if the software artifact is released independently of any existing community. In the latter case, much more resources are required long-term both in terms of managing the community, but also maintaining and leading the software development within the community. As highlighted by I10, *"we prefer contributing to existing communities because it is expensive taking on the role of a maintainer in a larger project"*.

Example [CaseOrg3]:   CaseOrg3 recognizes the costs implied by sharing a software artifact as OSS as a concern, but sees it from a long-term perspective. I15 asks, *"What is the need of the labor market? If there is a potentially positive outcome, then I think we are prepared to spend the money needed"*.

Mitigation strategy: Develop software artifacts as if they were intended to be released as OSS from start, separating commodity and differentiating functionality. Also, ask *"who*

is going to be the owner of this repo, and have you allocated the time to maintain it, and has your boss approved your time budget?" (I1).

D.4.2 CC11 - Modularity and architecture:

Description: In certain cases, it may be that the software artifact is too embedded in internal infrastructure, which makes it infeasible to modularize the artifact.

Concern [CaseOrg1,2&3]:   Technical feasibility to modularize and abstract software artifact.

Example [CaseOrg]:   I1 stresses that *"companies should start projects with the goal that it will be open one day, then they could architect it right from the start with generality and modules that can be contributed"*.

Mitigation strategy: Develop software artifacts as if they were intended to be released as OSS from start, separating commodity and differentiating functionality. An option may be to contribute the *"design document or the blueprint of the project itself so someone else can create it"*, i.e., documentation that in some way captures the knowledge from the underlying software artifact.

D.4.3 CC12 - Code alignment:

Description: By not contributing internally developed code that relates to a project, a misalignment between the internally and externally developed software may arise. A negative consequence may be that the organization unintentionally ends up on a fork that grows with time and creates unnecessary maintenance and patch-work.

Concern [CaseOrg1,2&3]:   Cost of maintaining internal fork. Misalignment between internal and external development may prevent or complicate future contributions.

Example [CaseOrg2]:   I8 explains it as, *"If you don't share, the community may take off in another direction. Then you will stagnate and come to a place that will be very hard to get back from"*.

Mitigation strategy: Keep internal fork of concerned OSS as close as possible to the community's.

D.5 Community-centric Complexities

D.5.1 CC13 - External interest:

Description: To contribute a software artifact to an existing OSS community, or to create a new community around it, there needs to be an external interest for the artifact.

Concern [CaseOrg1,2&3]:   Risk of contribution not being accepted or community not being established.

Example [CaseOrg1]:   I3 describes it as *"filling in a gap"*. I5 asks the question, *"Is there a general purpose part of this that I can see multiple teams inside my company taking advantage of?"*. External interest should, therefore, be investigated before proceeding with any contribution. In one example where CaseOrg1 ended up creating a new community, I6 explained, *"everyone needs to do it, and it is not really great agreement over what the right methodologies are"*.

Example [CaseOrg2]:   For *"existing communities that have been around for long you know, or at least get an indication, if it is going to be an uptake or not [of the contribution]"* (I7). Analyzing stakeholders' agendas within a community can further help. *"What's their strategy, whom do they have playing, what are they trying to get done, what*

chess moves are they making, and if so, which then informs what we contribute, when we contribute, and how strongly we need to be present" (I1).

Mitigation strategy: Investigate external interest and needs within concerned communities and industries, and consider the generality of the use-case that the software artifact solves.

D.5.2 CC14 - Influence in community:

Description: If the external interest within a community is weak, or if the community is heading in a different direction, it may be important to have influence on the concerned community's RE process in order to create traction and approval for the contribution, but also to be able to steer its development if it is accepted (Linåker et al. 2019a).

Concern [CaseOrg1,2&3]: Low level of influence in the community may prevent or complicate the contribution of a software artifact.

Example [CaseOrg2]: I7 highlights that *"It is important for [CaseOrg2] to pick up a leading role in certain communities that we value as strategically important and as a potential way to get an edge against competitors in those communities"*.

Concern [CaseOrg1,2&3]: Target foundation may require a governance model too open which may render in too large scope and loss of control.

Example [CaseOrg1]: I1 explains how CaseOrg1 reasoned about the creation of a separate community behind an internally developed Linux distribution, *"We could have contributed that code to the Linux Foundation or the Apache Software Foundation and have them make it a broader project. But we ended up creating our own foundation and collected all the [relevant stakeholders] together to contribute towards this platform that we created. In a way, it is the right way because we wanted to make sure that it served the need of our [main customers] and it didn't get too broad and become an embedded system that everyone uses... So we want to maintain influence and be in a controlling position, so we are one of three members of the steering committee"*.

Mitigation strategy: Build influence needed on concerned community's RE process. If e.g., the current level of influence is deemed not high enough to be able to contribute and steer the software artifact, one option is to create a new community where the community's governance structure can be tailored based on the focal organization's needs.

D.5.3 CC15 - Community health:

Description: Contributing to and engaging actively with a community is one way to support its health, i.e., ability to survive throughout time (Wahyudin et al. 2007), which is an important aspect if an organization is dependent on a specific OSS project (Linåker et al. 2019a).

Concern [CaseOrg1,2&3]: Not contributing may have a negative impact on the health of the OSS project.

Example [CaseOrg1]: I1 explain CaseOrg1's approach as *"We care about the health of the project, will it die because there are not enough contributors maintaining it? We cannot let that project die because we are using it and we would have to swap out the code and go to something else"*. I1 continues, *"We prefer to use something that is already there, and not reinvents the wheel, but if it is not going to be healthy, then we would want to be there just to create a vibrant community, not for influence, but for health"*.

Mitigation strategy: Monitor and analyze the health of concerned OSS communities.

References

ÅGerfalk PJ, Fitzgerald B (2008) Outsourcing to an Unknown Workforce: Exploring Opensurcing as a Global Sourcing Strategy. MIS Quarterly 32(2):385–409

Alspaugh TA, Scacchi W (2013) Ongoing software development without classical requirements. In: 21st IEEE International Requirements Engineering Conference, RE'13. IEEE, Rio de Janeiro pp 165–174

Andersen-Gott M, Ghinea G, Bygstad B (2012) Why do commercial companies contribute to open source software? Int J Inf Manag 32(2):106–117

Bosch J (2013) Achieving simplicity with the Three-Layer product model. Computer 46(11):34–39

Butler S, Gamalielsson J, Lundell B, Jonsson P, Sjöberg J, Mattsson A, Rickö N, Gustavsson T, Feist J, Landemoo S (2018) An investigation of work practices used by companies making contributions to established OSS projects. In: 40th International Conference on Software Engineering: Software Engineering in Practice, ICSE'18. IEEE, Gothenburg, pp 201–210

Caulkins JP, Feichtinger G, Grass D, Hartl RF, Kort PM, Seidl A (2013) When to make proprietary software open source. J Econ Dyn Control 37(6):1182x–1194

Chengalur-Smith I, Nevo S, Demertzoglou P (2010) An empirical analysis of the business value of open source infrastructure technologies. J Assoc Inf Syst 11(11):708

Chesbrough Henry, Appleyard MM (2007) Open innovation and strategy. Calif Manag Rev 50(1):57–76

Chesbrough H, Vanhaverbeke W, West J (eds) (2014) New Frontiers in Open Innovation. Oxford University Press, November

Dahlander L, Magnusson MG (2005) Relationships between open source software companies and communities: Observations from Nordic firms. Res Policy 34(4):481–493

Dahlander L, Wallin MW (2006) A man on the inside Unlocking communities as complementary assets. Res Policy 35(8):1243–1259

Dahlander L, Magnusson MG (2008) How do firms make use of open source communities? Long Range Plan 41(6):629–649

DaSilva CM, Trkman P (2014) Business model: What it is and what it is not. Long Range Plan 47(6):379–389

Deodhar SJ, Saxena KBC, Gupta RK, Ruohonen M (2012) Strategies for software-based hybrid business models. J Strat Inf Syst 21(4):274–294

Duc AN, Cruzes DS, Hanssen GK, Snarby T, Abrahamsson P (2017) Coopetition of software firms in open source software ecosystems. In: Ojala A, Olsson HH, Werder K (eds) Software business. Springer International Publishing, Cham, pp 146–160

Easterbrook S, Singer J, Storey M-A, Damian D (2008) Selecting empirical methods for software engineering research. In: Guide to advanced empirical software engineering. Springer, pp 285–311

Ernst N, Murphy GC (2012) Case studies in just-in-time requirements analysis. In 2nd International Workshop on Empirical Requirements Engineering, empiRE'12. IEEE, Chicago pp 25–32

Flyvbjerg B (2007) Five misunderstandings about Case-Study research. In: Qualitative research practice. SAGE, concise paperback edition, pp 390–404

Ghaisas S, Rose P, Daneva M, Sikkel K, Wieringa RJ (2013) Generalizing by similarity: Lessons learnt from industrial case studies. In: Proceedings of the 1st International Workshop on Conducting Empirical Studies in Industry, CESI '14. IEEE Press, Piscataway, pp 37–42

Gorschek T, Garre P, Larsson S, Wohlin C (2006) A model for technology transfer in practice. IEEE Softw 23(6):88–95

Hartmann H, Bosch J (2016) Towards a multi-criteria decision support method for consumer electronics software ecosystems. J Softw Evol Process 28(6):460–482

Haruvy E, Sethi SP, Zhou J (2008) Open source development with a commercial complementary product or service. Prod Oper Manag 17(1):29–43

Hauge Ø, Ayala C, Conradi R (2010) Adoption of open source software in software-intensive organizations - a systematic literature review. Form Softw Technol 52(11):1133–1154

Henkel J (2006) Selective revealing in open innovation processes: The case of embedded linux. Res Policy 35(7):953–969

Henkel J (2008) Champions of revealing-the role of open source developers in commercial firms. Dustrial Corp Chang 18(3):435–471

Henkel J, Schöberl S, Alexy O (2014) The emergence of openness: How and why firms adopt selective revealing in open innovation. Res Policy 43(5):879–890

Hevner AR, March ST, Park J, Ram S (2004) Design science in information systems research. MIS Quart 28(1):75–105

Höst M, Orucevic-Alagic A (2011) A systematic review of research on open source software in commercial software product development. Form Softw Technol 53(6):616 – 624

Iivari N, Hedberg H, Kirves T (2008) Usability in company open source software context - initial findings from an empirical case study. In: Russo B, Damiani E, Hissam S, Lundell B, Succi G (eds) Open source development, communities and quality. Springer, Boston, pp 359–365

Jansen S, Finkelstein A, Brinkkemper S (2009) A sense of community: A research agenda for software ecosystems. In: 31st International Conference on Software Engineering - Companion Volume, ICSE'09. IEEE, Vancouver, pp 187–190

Jansen S, Brinkkemper S, Souer J, Luinenburg L (2012) Shades of gray Opening up a software producing organization with the open software enterprise model. J Syst Softw 85(7):1495–1510

Kilamo T, Hammouda I, Mikkonen T, Aaltonen T (2012) From proprietary to open source - Growing an open source ecosystem. J Syst Softw 85(7):1467–1478

Kittlaus H-B, Fricker SA (2017) Software Product Management: The ISPMA-Compliant Study Guide and Handbook. Springer

Kort PM, Zaccour G (2011) When should a firm open its source code: a strategic analysis. Prod Oper Manag 20(6):877–888

Kuriakose J, Parsons J (2015) How do Open Source Software (OSS) developers practice and perceive requirements engineering? An empirical study. In: 5th International Workshop on Empirical Requirements Engineering, empiRE'15. IEEE, Ottawa, pp 49–56

Laurent P, Cleland-Huang J (2009) Lessons learned from open source projects for facilitating online requirements processes. In: Glinz M, Heymans P (eds) Requirements engineering: Foundation for software quality. Springer, Berlin, pp 240–255

Linåker J, Regnell B, Munir H (2015) Requirements engineering in open innovation: a research agenda. In: Proceedings of the 2015 International Conference on Software and System Process, ICSSP'15. ACM, pp 208–212

Linåker J, Munir H, Wnuk K, Mols C-E (2018) Motivating the contributions: An open innovation perspective on what to share as open source software. J Syst Softw 135:17–36

Linåker J, Regnell B, Damian D (2019a) A community strategy framework - how to obtain influence on requirements in meritocratic open source software communities? Information and Software Technology

Linåker J, Regnell B, Damian D (2019b) A method for analyzing stakeholders' influence on an open source software ecosystem's requirements engineering process. Requirements Engineering

Lindman J, Juutilainen J-P, Rossi M (2009) Beyond the business model Incentives for organizations to publish software source code. In: Boldyreff C, Crowston K, Lundell B, Wasserman AI (eds) Open source ecosystems: Diverse communities interacting. Springer, Berlin, pp 47–56

Lundell B, Lings B, Lindqvist E (2010) Open source in Swedish companies: where are we? Inf Syst J 20(6):519–535

Lundell B, Lings B, Syberfeldt A (2011) Practitioner perceptions of Open Source software in the embedded systems area. J Syst Softw 84(9):1540–1549

Mäenpää H, Mäkinen S, Kilamo T, Mikkonen T, Männistö T, Ritala P (2018) Organizing for openness: six models for developer involvement in hybrid OSS projects. J Internet Serv Appl 9(1):17

Morgan L, Finnegan P (2014) Beyond Free software: An exploration of the business value of strategic open source. J Strat Inf Syst 23(3):226–238

Munir H, Wnuk K, Runeson P (2016) Open innovation in software engineering: a systematic mapping study. Empir Softw Eng 21(2):684–723

Munir H, Låker J, Wnuk K, Runeson P, Regnell B (2018a) Open innovation using open source tools: a case study at sony mobile. Empir Softw Eng 23(1):186–223

Munir H, Runeson P, Wnuk K (2018b) A theory of openness for software engineering tools in software organizations. Form Softw Technol 97:26–45

Nagle F (2018) Learning by contributing: gaining competitive advantage through contribution to crowd-sourced public goods. Organ Sci 29(4):569–587

Nguyen-Duc A, Cruzes DS, Terje S, Abrahamsson P (2019) Do Software Firms Collaborate or Compete? A Model of Coopetition in Community-initiated OSS Projects. e-Inform Softw Eng J 13(1):37–62

O'Mahony S (2007) The governance of open source initiatives: what does it mean to be community managed? J Manag Govern 11(2):139–150

Olsson HH, Bosch J (2017) From ad hoc to strategic ecosystem management: The Three-Layer Ecosystem Strategy Model (teLESM). J Softw Evol Process 29(7):e1876

Riehle D (2011) Controlling and steering open source projects. Computer 44(7):93–96

Riehle D (2012) The single-vendor commercial open course business model. In: Formation systems and e-business management, 10(1):5–17

Robson C (2011) Real world research, vol 3. Wiley, Chichester

Runeson P, Höst M, Rainer A, Regnell B (2012) Case study research in software engineering - guidelines and examples. Wiley, New York

Scacchi W (2002) Understanding the requirements for developing open source software systems. In: Software, IEE proceedings-, vol 149. IET, pp 24–39

Schaarschmidt M, Walsh G, von Kortzfleisch HFO (2015) How do firms influence open source software communities? a framework and empirical analysis of different governance modes. Form Organ 25(2):99–114

Seaman CB (1999) Qualitative methods in empirical studies of software engineering. IEEE Trans Softw Eng 25(4):557–572

Shahrivar S, Elahi S, Hassanzadeh A, Montazer G (2018) A business model for commercial open source software A systematic literature review. Form Softw Technol 103:202–214

Shaikh M, Henfridsson O (2017) Governing open source software through coordination processes. Form Organ 27(2):116–135

Spijkerman Z, Jansen S (2018) The open source software business model blueprint: a comparative analysis of 10 open source companies. In: Proceedings of the International Workshop on Software-intensive Business: Start-ups, Ecosystems and Platforms, vol 2018, pp 128–145

Stam W (2009) When does community participation enhance the performance of open source software companies? Res Policy 38(8):1288–1299

Stuermer M, Spaeth S, Von Krogh G (2009) Extending private-collective innovation: a case study. R&D Manag 39(2):170–191

Syeed M, Lindman J, Hammouda I (2017) Measuring perceived trust in open source software communities. In: Balaguer F, Di Cosmo R, Garrido A, Kon F, Robles G, Zacchiroli S (eds) Open source systems: Towards robust practices. Springer International Publishing, Cham, pp 49–54

Van der Linden F, Lundell B, Marttiin P (2009) Commodification of industrial software A case for open source. IEEE Softw 26(4):77–83

Ven K, Mannaert H (2008) Challenges and strategies in the use of open source software by independent software vendors. Form Softw Technol 50(9):991–1002

Wahyudin D, Mustofa K, Schatten A, Biffl S, Min Tjoa A (2007) Monitoring the "health" status of open source web-engineering projects. Int J Web Inf Syst 3(1/2):116–139

Weikert F, Riehle D (2013) A model of commercial open source software product features. In: Herzwurm G, Margaria T (eds) Software business. From physical products to software services and solutions. Springer, Berlin, pp 90–101

West J (2003) How open is open enough?: Melding proprietary and open source platform strategies. Res Policy 32(7):1259–1285

West J, O'Mahony S (2005) Contrasting Community Building in Sponsored and Community Founded Open Source Projects. In: Proceedings of the 38th Annual Hawaii International Conference on System Sciences, HICSS'05. IEEE, Big Island pp 196–196

West J, Gallagher S (2006) Challenges of open innovation: the paradox of firm investment in open-source software. R&D Manag 36(3):319–331

West J (2007) Value capture networks in open source vendor strategies. In: Value Proceedings of the 40th Annual Hawaii International Conference on System Sciences, HICSS'07. IEEE, Waikoloa pp 176–176

West J, Wood D (2008) Creating and Evolving an Open Innovation Ecosystem: Lessons from symbian ltd available at SSRN 1532926

Wieringa RJ (2014) Design science methodology for information systems and software engineering. Springer, Berlin

Wnuk K, Pfahl D, Callele D, Karlsson E (2012) How can open source software development help requirements management gain the potential of open innovation: An exploratory study. In: Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM'12. ACM, New York pp 271–280

Zhou M, Mockus A, Ma X, Zhang L, Mei H (2016) Flow and retention in oss communities with commercial involvement: a case study of three hybrid projects. ACM Trans Softw Eng Methodol (TOSEM) 25(2):1–29

Ziegler N, Gassmann O, Friesike S (2014) Why do firms give away their patents for free? World Patent Inf 37:19–25

Affiliations

Johan Linåker[1] · Björn Regnell[1]

   Björn Regnell
   bjorn.regnell@cs.lth.se

[1]   Box 118, SE-221 00 Lund, Sweden