Leiden University
ICT in Business and the Public Sector

Contributing to open-source software projects:

Risks and risk mitigation strategies for companies

Okke Moison
s1814702
April 23, 2024

First supervisor: Dr. W. Heijstek
Second supervisor: Drs. J.B. Kruiswijk

MASTER THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

Introduction

Open-source software projects are widely used by companies. However, those companies do not always contribute to these projects. Even companies that use a project extensively and develop software to improve it do not always contribute that software. While prior research has found benefits and challenges to contributing, no research has been performed specifically looking at the risks and risk mitigation strategies for contributing to open-source software projects that are not owned or controlled by the company that contributes. Addressing this gap in the literature can give companies the necessary understanding of these risks, strategies to mitigate those risks and the tools to contribute to open-source projects with confidence.

Methodology

Fourteen semi-structured interviews have been conducted with industry experts. These experts range from software developers who work with and contribute to open-source projects to managers who make decisions on open-source policy within their company. They work with different types of open-source software and are employed at companies both small and large. In accordance with the grounded theory approach, the interview transcripts have been coded in two rounds to generate the theory presented in the results.

Results

Nine risk categories emerged from the interviews: brand image damage, rejected contributions, financial, intellectual property, licenses, security, sustainability, business customers and not contributing. These categories are each divided into sub-risks. For most of these, one or more risk mitigation strategies have been found. These are explained in detail, and for every risk category its risks and corresponding mitigation strategies have been visualised.

Discussion

Not contributing code that has been developed for open-source projects and instead maintaining it internally, emerged as one of the biggest risks surrounding open-source development. Following this, the risk that contributions by the company are rejected also has a high impact. Other risks had less impact, or were found to have risk mitigation strategies that reduced those risks significantly. Risk mitigation strategies that were found to have an impact on multiple risks, or on overall risk reduction, were clear communication internally and with the open-source communities, creation of policy supported by tooling and education of employees and implementing an internal review process of contributions.

Conclusion

In practice, the results can be used to draft policy for contributing to open-source software projects. The general mitigation strategies can form the basis for this. With the most impactful risk categories in mind, along with the fact that not contributing code that has been developed usually carries a large risk as well, a well-rounded policy can be developed. Other risks, while potentially harmful still, have a lower impact or apply to certain companies to a lesser extent, and should be considered on a case-by-case basis.

Future work could focus on understanding the exact costs of developing code for open-source projects and not contributing it and on how to build influence in communities. Specific risks that are interesting for future research are the sustainability of small projects and the reasons that open-source maintainers reject contributions.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

Open-source software (OSS) is an important part of the software landscape. Popular projects, such as Linux, Python and Mozilla Firefox, have gathered large user bases. Although OSS started as a fringe movement in the software community, it has become a professionalised area over time (Fitzgerald, 2006).

1.2 Consuming open-source software

For companies, using open-source software offers some valuable benefits. The fact that OSS is often free to use and that users can read, change and redistribute the source code are some of the obvious benefits these products provide (Morgan & Finnegan, 2007; Open Source Initiative, 2007). Other, perhaps less obvious, benefits of consuming mature OSS products include: Potentially higher reliability and security when compared to proprietary software (AlMarzouq et al., 2005; Morgan & Finnegan, 2007) and increased collaboration and innovation (Andersen-Gott et al., 2012; Linåker & Regnell, 2020; Morgan & Finnegan, 2007; Munir et al., 2016). Additionally, research by Nagle (2019) found that a company can increase its productivity by increasing the amount of free OSS it uses, if those projects complement the existing software ecosystem of the company. However, companies do need the internal capabilities to make use of the benefits open-source software projects offer (Morgan & Finnegan, 2007; West & Gallagher, 2006).

There are some risks and disadvantages to consuming OSS. Whereas proprietary software vendors often offer technical support as part of a software purchase, open-source projects rarely provide a way to acquire support (AlMarzouq et al., 2005; Morgan & Finnegan, 2007). OSS projects may not have a roadmap or a clear release schedule, making it difficult to understand the development direction of the software (Morgan & Finnegan, 2007; Munir et al., 2016).

1.3 Contributing to open-source software

One of the unique characteristics of open source software is that it is possible to change the source code. In fact, it is possible to change the code and then donating this change to the project. This is called "contributing". If the person or organisation in charge of the project find the contribution useful and according to their quality standards, it may be integrated in the source code of the project. This has several benefits for a company. For example, any contribution that is accepted by an open-source project will be maintained by other developers working on that project (Andersen-Gott et al., 2012; Kendall et al., 2016; Munir et al., 2016). Additionally, by contributing a company can become a more attractive employer and it can scout new talent in open-source communities that it is active in (Lerner et al., 2006; Linåker & Regnell, 2020).

1.4 Problem Statement

Contributing to open source projects does comes with risks and challenges. These risks include, among other things, intellectual property loss, misalignment between the project direction and the company needs (Linåker & Regnell, 2020; Munir et al., 2016), and additional investment of resources (Butler et al., 2019). Although there has been research into what risks exist around contributing to open-source software, there has been little research into ways to mitigate those risks. Linåker and Regnell (2020) do propose mitigation strategies for the risks they found, but this is limited to at most a few sentences per risk. As they note in their conclusion: more validation and generalisation is needed to create a framework on this topic. Additionally, as Butler et al. (2019) mention, most research focuses on a scenario where the company has a controlling interest in - or even ownership of - the project they contribute to. This means they could more easily reduce some risks, as they have more influence over the project. On the other hand there is limited literature about the scenario where a company does not have this influence. In this case a company contributes to a project that they do not own and where they do not have a controlling interest in the project. Although it is possible that these companies have an important position in the project and can exert influence, they cannot steer the direction of the project by themselves (Butler et al., 2019; Linåker & Regnell, 2020). Research that does take this scenario into account (Linåker & Regnell, 2020; Lundell et al., 2021; Munir et al., 2016) combine it with the scenario where the company does have a controlling influence, so it does not look into it in isolation. Butler et al. (2019) do do that, but they focus mostly on work practices developers use when they contribute, not on risks and challenges.

Here we find a gap in the literature: there is little knowledge about risk mitigation strategies for companies that contribute to projects that they do not own or control.

1.5 Objectives

The objective of this thesis is to create an understanding about risk mitigation strategies for contributing to open-source software projects. As mentioned before, specifically those projects that are not owned by the company itself will be considered. If such strategies are found, companies that work with open-source software can decrease the risk of contributing. Consequently, they may contribute code to projects that would otherwise have been kept proprietary.

To reach this objective, the research question will be:

How can companies mitigate the risks of contributing to open-source software projects they do not control?

1.6 Thesis Overview

In this chapter, the topic was introduced, the objectives were explained and the research question was presented. Chapter 2 will cover earlier research into this topic to support the research question and show where it fits in the field. Additionally this chapter will feature a more in-depth look into the benefits and risks in OSS. Chapter 3 will cover the method through which the research question will be answered. Chapter 4 will show the results of the performed research and these discoveries will be discussed further in Chapter 5. Finally, the thesis will be concluded in Chapter 6.

2 Related Work

In this section, an overview is given on how open-source software works in practice. Then, the benefits and risks around open source software will be discussed. After that the benefits of contributing to OSS are explained. Finally, the risks around contributing, mentioned first in chapter 1.4, are discussed in more detail.

2.1 Open-source in practice

An open-source project can be started by anyone, whether that is a single person, a group of people or a company. The ones who started the project are usually the core developers (or "maintainers") for a project: they have final say about the source code and the direction of development. If the project is useful to others, a community can form around it. All users – even those who only use the software and do not contribute – are seen as members of the community. Some of those, the active users, will report bugs and issues with the software and suggest new features. Some of the active users may become developers if they have the technical skills for it. They fix bugs and create new features, which they make available to the core developers. The core developers can then decide whether or not to add those contributions to the source code. Some developers may move on to become core developers in time (AlMarzouq et al., 2005).

Although the details may differ for each project, the contribution and release process is often similar to the following (AlMarzouq et al., 2005; Butler et al., 2019):

1. A request is made for a new feature. It is discussed in the community and by the core developers.
2. When a consensus about the requirements is reached, a developer – potentially the requester, but not necessarily – implements the feature along with tests and other requirements set by the project rules.
3. The developer opens a pull request so the core developers know the feature is implemented. They will then review and test the code.
4. If changes need to be made, steps 2 and 3 will be repeated until the code is deemed acceptable by the core developers. If so, it is merged with the development build of the software.
5. Finally, the feature is included in the next release package of the software, along with any other new features or bug fixes added to the development build.

Bug fixes are usually handled in a similar way, but usually require less discussion.

2.2 Benefits of Contributing

There are many benefits to contributing to open-source software projects. Understanding these benefits is helpful when considering the risks: without a reason to contribute in the first place, looking at the risks is not very useful.

2.2.1 Reduced Costs

When a piece of software is contributed to a project, it may be added to the main code base. If this happens, the company that contributed this piece of software is no longer responsible for maintaining it. From that point on, the core developers will maintain it (Andersen-Gott et al., 2012; Butler et al., 2019; Kendall et al., 2016). On the other hand, when a feature is kept private, the maintenance efforts and new releases of the project from the community cannot take that feature into account. Consequently, any new release may break compatibility with that feature, incurring unexpected cost of updating the code to fit with the new release. If instead this feature had been contributed and integrated into the project, it would have been compatible with the new release as it would have been part of the code base during the development and maintenance effort by the community (Butler et al., 2019; Kendall et al., 2016).

2.2.2 Complementary Services and Products

Although it is hard to generate revenue from an open-source project directly – the source code is freely available after all – a strategy that can be employed is to offer complementary products or services. A common example is to offer consultancy or technical support services for a project. A company could also decide to develop extensions or plug-ins for an OSS project, which – if the open-source license allows for it – it would be able to keep proprietary and sell. Any other company can offer these services as well, with a low barrier to entry as they have access to the same information. An advantage in knowledge and experience gained through contributing to the project can lead to a competitive advantage over others (Andersen-Gott et al., 2012; Lerner et al., 2006)

2.2.3 Leverage External Knowledge and Talent

Generally, when code is contributed to a project it is reviewed by other developers in the community (Butler et al., 2019; Fitzgerald, 2006). This can be valuable feedback, because core developers are experts on the project. This feedback is not available if code is not contributed. Additionally, anything that has been contributed is open to others in the community to extend and improve upon. This can lead to more value, without additional cost or effort (Andersen-Gott et al., 2012).

2.2.4 Employees Satisfaction and Recruitment

Linåker and Regnell (2020) found that contributing to OSS projects improves branding towards employees. Both for current employees and for attracting new employees, being able to contribute to open-source projects is a benefit. Being active in open-source communities is not only useful to attract employees, it can also be a useful tool for scouting of new talent (Lerner et al., 2006).

For employees, there are multiple benefits to contributing to open-source software. Employees like it when the code they worked on will stay useful and in use. Contributing it to an open-source project ensures the community can take advantage of it, whereas it may be deprecated or removed from the product portfolio of a company when it is no longer deemed useful (Kendall et al., 2016). The fact that open-source contributions are usually peer-reviewed and the contributor receives feedback, offers developers the opportunity to learn and improve their skills, which motivates them (Oreg & Nov, 2008). Another motivation for contributing is the opportunity to build a reputation in the open-source community (Oreg & Nov, 2008). Finally, employees may want to contribute out of altruism or because they feel feel it is the right thing to do to give back to the open-source projects they use for free (Linåker & Regnell, 2020; Oreg & Nov, 2008).

2.2.5 Increase Influence

Engaging with the community and contributing regularly can increase the influence of a company in the project. That company may then influence the direction of the project and the priority of feature development (AlMarzouq et al., 2005; Kendall et al., 2016).

2.2.6 Increased Learning

Companies can learn from being are actively involved in the development of open source projects. Besides the technical skills developers can pick up, a deeper understanding of the software itself can be acquired (AlMarzouq et al., 2005).

2.3 Licenses

Open-source software projects almost always have a licence attached to them. This license prescribe what users can and cannot do with the source code. Although an depth explanation of all licences that are in use is outside the scope of this project, the most common ones will be discussed. The different types of open-source licenses can be divided into three categories: highly restrictive licenses, restrictive licenses and permissive licenses (Lerner & Tirole, 2005).

2.3.1 Highly Restrictive Licenses

One of the earliest open-source licenses is the GNU General Public License (GPL). This license is known for its stipulation that the source code should be made available for any distribution, and that any derived work may only be distributed under the same license (De Laat, 2005).

This first clause, that the source code should be made available for any distribution of the software under the license, makes GPL restrictive (Lerner & Tirole, 2005). This feature of open-source licensing is sometimes called "copyleft", as a counterpart to copyright. (De Laat, 2005; Lerner & Tirole, 2005). It should be noted however, that this does not mean that all changes should be made publicly available: only the recipients of the software have to receive the source code (Henkel, 2006). This means that changes that are distributed only for internal use in a company can be kept private.

The second clause, that derived works should be distributed only under the same license, makes GPL a highly restrictive license (Lerner & Tirole, 2005). This clause means in effect that any changes made to the source code under GPL that is then distributed, should be distributed under GPL as well (De Laat, 2005). It is sometimes referred to as a "viral" license. The reach of this feature is broad: it does not only cover changes to or extensions of the original program, it also includes all programs that link to the software (Lerner & Tirole, 2005). That means that under the terms of the license the source code of proprietary software can be forced to be revealed if it links with a library under GPL.

2.3.2 Restrictive Licenses

Restrictive licenses are licenses that do have copyleft clauses, but that have a weaker viral clause, or no viral clause at all (Lerner & Tirole, 2005). Restrictive and highly restrictive licenses are sometimes called weak and strong copyleft licenses respectively. Two well-known licenses in this category are Lesser GPL (LGPL) and the Mozilla Public License (MPL).

The LGPL is a variation on the GPL. It permits the distribution of software that links to LGPL software without it needing to carry an LGPL license itself (De Laat, 2005; Lerner & Tirole, 2005).

The MPL goes a step further: while modifications to source files under the MPL should be made available under similar copyleft rules as the (L)GPL licenses, not all modifications have to be opened. As long as modifications are kept in separate files (and the original source code under the MPL license is made available) the separated files do not have to be licensed under MPL and can be kept proprietary (De Laat, 2005).

2.3.3 Permissive Licenses

Other licenses, that are neither viral nor copyleft, are called unrestrictive or permissive licenses (Lerner & Tirole, 2005). The most common ones are the Berkeley Software Distribution License (BSD), the Apache License and the MIT License (De Laat, 2005; Lundell et al., 2021).

These licenses have in common that they allow any use of the software, including but not limited to modifying, distributing and selling the software, without the copyleft or viral conditions. This means that the source code of software licensed under a permissive license can be used in a proprietary product without having to publish any source code (De Laat, 2005).

2.4 Risks of Contributing

As discussed in chapter 1.4, contributing to open source software projects comes with risks. There may be differences in the type or severity of risks between OSS projects that are owned or controlled by the company, versus projects where the company does not have such control. Logically, having influence in a project reduces some risks, while owning a project may come with additional risks. As this thesis is concerned with the scenario where a company does not control a project, this will be the focus of this chapter. However, differences in severity and types of risks – if known – will be mentioned.

2.4.1 Intellectual Property

There are multiple risks concerning intellectual property. One risk is that contributing some feature damages the competitive advantage of the company, because competitors can use it as well. Deciding what can be contributed and when is thus an important part of the contribution process (Linåker & Regnell, 2020; Munir et al., 2016). Giving away sensitive or patentable intellectual property is a risk as well. Others may abuse this by starting lawsuits, so building a defensive patent portfolio is important to protect the company against that (Linåker & Regnell, 2020).

2.4.2 Community

There is a risk that the direction the company wants to go in with the open source project does not align with the direction the community wants the project to go. Consequently, features the company develops may not get accepted and integrated in the project (Linåker & Regnell, 2020). This forces the company to maintain an internal fork or to abandon the features they want.

For companies that do not only contribute to projects of others, but also release their own projects as open-source, there is a risk that no community is formed. Of course a company can invest in garnering attention for the project, and if the software is useful and of high quality this risk is reduced, but there is always a chance that there are not enough people interested in contributing (AlMarzouq et al., 2005). This is not of immediate concern of companies that only contribute to projects and do that do not start their own.

2.4.3 Brand Image

Contributing to OSS may pose a risk to the brand image of the company. It is possible that software that they released is used for nefarious purposes, as

any use is allowed under open source licenses, which may reflect poorly on the company (Linåker & Regnell, 2020). This risk is higher for projects that the company owns, as its name is more clearly attached to it.

2.4.4 Security

As with any software development process, introducing security problems is a risk. However, because the source code is then contributed and opened, those issues may be exposed. Additionally, it is possible to expose valuable data by accidentally sharing, for example, a database key (Linåker & Regnell, 2020).

2.4.5 License

There are many different open-source licenses and some of them can be complicated, so not complying to a license is a risk (Linåker & Regnell, 2020). As discussed in Chapter 2.3, compliance to licenses often revolves around distribution of software and whether or not source code should then be opened as well. Consequently, the risk is using code from an OSS project under a (highly) restrictive license when it is intended to keep the newly developed software proprietary: the proprietary code can then be forced to be distributed as well.

2.4.6 Financial

There may be more costs involved when contributing to open-source software when compared to keeping code internal. Open-source projects expect code that fits with their source code, so anything that is contributed should be generalised (AlMarzouq et al., 2005; Linåker & Regnell, 2020). Additionally, there may be multiple rounds of code review from the project before a contribution is accepted, leading both to delays in integration and more development hours (Butler et al., 2019). Adjusting the development process of the company to this review process may be required, leading to even higher costs (AlMarzouq et al., 2005).

2.4.7 Not Contributing

Although contributing may come with additional costs, not contributing brings the risk of additional costs as well. When a feature is contributed and accepted, it is integrated into the source code of the project and will be included in the maintenance efforts of the community. This means that any new releases will take this feature into account as well. On the other hand, if it is not contributed, the task of maintaining falls on the company. Additionally, if the open source project releases a new version, the company may have to expend extra effort to make sure the feature still works with the new release. All this can lead to an increase in complexity and cost for the company (Butler et al., 2019; Linåker & Regnell, 2020).

Additionally, if there are not enough contributors for a project, the health of the community may decline. Not contributing is thus a risk to the health of the project (Linåker & Regnell, 2020).

3 Method

3.1 Qualitative Research Method

To answer the research questions posed in chapter 1.5, a qualitative research approach was chosen. Specifically, the grounded theory method will be used to perform this research, with data collection through semi-structured expert interviews. The grounded theory approach aims to construct a substantive theory which is applicable in practice. Through analysis of the data – in this case by coding the interviews and comparing the data found in different interviews – this theory emerges (Merriam & Tisdell, 2015).

3.2 Data Collection

Semi-structured interviews are a data collection method to gather structured data, while allowing for flexibility to divert from the script for in-depth discussion with interviewees. This fits the goal of this study well, since the subject of risk mitigation is a complex issue (Kallio et al., 2016).

A case study approach can provide more in-depth data compared to a semi-structured interview approach. However, interviews have the advantage of gathering a broader view on the risks and mitigation strategies that exist, as more different companies can be reached.

Another option for data collection would have been through a survey. Although this would allow for more participants and thus possibly an even broader view on the subject, it may also lead to more shallow answers. Here the option to easily ask follow-up questions is a benefit of the interview format over a survey format. It ensures that answers obtained are in-depth enough to provide useful data.

Considering these arguments, semi-structured interviews were chosen for the method of data collection. For this topic it provides a good balance between a broad view and in-depth data.

The interview guide and its justification can be found in Appendix A.

3.3 Participant Selection

To gather valuable data, selection criteria for participants were defined as follows: Participants should contribute to open-source software projects not controlled by their company, or they should be involved with such projects on the management or policy level. Participants who work directly with open-source are expected to understand the operational risks of contributing well. Those who are involved with contributing to open-source on the management or policy level are expected to have knowledge of business risks involved with OSS.

Participant selection was done through random sampling. Two strategies were employed to source participants. Discord – a popular social media and communication platform – hosts multiple servers for open-source projects [1].

[1] https://discord.com/open-source

Here, developers and users can discuss the project, talk about new features and report bugs they encounter. In these servers a message was sent explaining the research goal and asking for participants. Only servers for projects with business use cases were approached. People who were interested were asked about their fit to the selection criteria and invited for an interview if they did.

The second approach was through LinkedIn messages. Here, a search was performed with the terms "open source" and "open source software". From the results, persons with job titles related to open-source software were approached to join for an interview. Similar to the approach through Discord, their fit to the selection criteria was assessed before an interview invite was sent.

Additionally, snowball sampling was attempted, but yielded no results.

3.4 Coding

Coding of the interview transcripts will be done in two rounds. In the first round, open coding is applied to mark as much potentially interesting data as possible. During this process, categories and sub-categories will emerge. Anything that may be of value is coded and categorised. Afterwards, selective coding is applied to refine the categories and sub-categories. Only those categories useful for development of the theory remain. This way, no useful data is missed, while leaving only the interesting data for theory development.

4 Results

4.1 Chapter overview

In this chapter, the results of the applied method will be discussed. First, information on the interviews will be given, followed by a short explanation of different ways the companies that the participants work for contribute. Then each risk and its mitigation strategies are presented, and finally the overall results will be discussed.

4.2 Interviews and Coding

Fourteen interviews were performed, with an average length of 45 minutes. In Table 1 relevant information about the interviewees is shown.

In the coding process of the interviews that were conducted, nine categories of risks were found. The code book including code frequency can be found in Appendix B.

4.3 Contribution Strategies

In general, when contribution to open-source software is discussed, it is implied that this entails new features or bug fixes. However, there are different ways to contribute. Reporting bugs that are found in the project is one way to do that. Especially if a company does not have the time or knowledge required to fix a bug by themselves, an extensive bug report is a valuable way to help out maintainers.

Another way to contribute without code is with documentation. Whether it is fixing mistakes, updating the text for new releases or including details that are missing, documentation contributions can be done by anyone with an understanding of the software.

Although open-source software is usually free, people do spend time and resources developing it. Some of this work is done as part of a paid job for a company, but most work – especially on smaller projects – is still done voluntarily. Supporting developers of projects with grants or sponsorships helps these projects grow and persist. Joining or hosting events and conferences centred around an open-source project or open-source in general is a way to contribute as well. Financial support could also be used as an incentive for open-source developers to work on features that the company might need.

Interview	Job title	Company	Main OSS Development	Source
1	Senior back-end engineer	Software consultancy	Various	Discord
2	Software developer	Software firm	Web development software	Discord
3	Technical lead	Embedded software firm	Embedded Linux	Discord
4	Owner & software developer	Startup software consultancy	iOS app development software	Discord
5	Owner & consultant	Startup software consultancy	iOS app development software	Discord
6	Senior software engineer	Open-source software firm	Linux and LDAP	Discord
7	Lead contributor support	Open-source software firm	DevOps software	Discord
8	Senior software engineer	Open-source software firm	Web development software	Discord
9	Open-source software engineer	Software firm	Various	LinkedIn
10	Open-source engineer	Open-source software firm	App development software	LinkedIn
11	Engineering manager	Big Tech firm	Linux based operating system	LinkedIn
12	Open-source engineer	Open-source software firm	Database software	LinkedIn
13	Software consultant	Freelance	Various	LinkedIn
14	OSPO program manager	Big Tech firm	Various	LinkedIn

Table 1: Interviewee descriptions

If a company wants a certain feature added to the project, but does not want to develop it themselves, they could hire a third party to do so. The third party can then contribute the feature to the project. It is also possible for a company to hire a third party for development and then contribute the features themselves.

If a company is an active contributor to a project, there are ways to increase involvement in a project. As each contribution has to be reviewed, a company can choose to get involved with this process and review the code of other contributors. It can also help in other maintenance tasks that are necessary in large software projects.

Another way to get involved is by engaging with the community. This can involve discussion about the direction of the project and feature and release planning. It may be desirable to attain an official position as core developer to exert influence in these discussions. Engaging with the community can also involve offering technical support and answering question from other developers working on the project, or from users who need help understanding how the software works.

4.4 Brand Image Damage

Contributing to open-source software projects can lead to damage to the brand image of the company. There are two aspects to this risk. The first is conduct of employees in the community and the quality of their work. The second aspect is on the policy side. It concerns the types of projects that the company decides to contribute to, or whether it decides not to contribute.

Because work and communication in open-source is public, there is a risk that the brand image is damaged by unprofessional conduct from employees. As employees usually contribute with an account that is connected to their company, or at least carries their own name, it is trivial to connect them to the company they work for. So if an employee does something that is not acceptable it can reflect poorly on the company. Five interviewees saw this as a risk. There are different ways in which this risk can occur. As the discussion in open-source communities is public, any comments that are made by an employee can lead to damage to the company brand:

> *"You do need to be respectable and professional with people, especially within open-source because those discussions are even more public."*

As one interviewee noted, improperly engaging with questions in the community or on the mailing list of projects can pose a risk as well. Incorrect or incomplete answers reflect poorly on the employee and thus on the company:

> *"I have seen people that just want exposure and they just answer whatever pops into their head, just for their name to be out there, but I try to be careful with that because I want to maintain our expertise level."*

Four interviewees talked about the importance of professional communication in open-source communities. Creating awareness among employees that they have a public facing role where they are seen as representatives of the company can help in this aspect:

> *"(...) and just trying to make more people aware that this is a thing: that you represent yourself, that you represent your company in these spaces and you want to ensure that you are behaving in line with the expectations set by the community."*

Employees might not have knowledge of the specific code of conduct set by communities where developers are active or even the general expectations in open-source. Offering education on these topics can then reduce the risk of misconduct in open-source communities:

> *"The best way as a company to reduce that risk is to educate employees how to behave publicly, it is not really different than public PR training or speaker training."*

Similarly, four interviewees found that low quality contributions or contributions that introduce security issues can be a risk factor to the company. Open-source projects expect contributions to be of a certain standard and often have requirements around style and the process of contributing as well. Consequently, not delivering contributions that are up to standard can lead to a poor brand image in the community. However, four other interviewees did not agree with that. They found that low quality contributions – at least in some communities – are an expected part of open-source development. In their experience, the open-source communities they worked with were very lenient and understanding about mistakes. Contributions to projects that are of poor quality are rejected without it reflecting poorly on the company:

> *"Some [communities] feel like: everyone makes mistakes, it is not really an issue."*

One interviewee had not found any issues with personal mistakes being connected with the company, as they found that open-source communities only looked at the individual:

> *"One thing that I really like in the open source community is that they look to you as individuals, not as a company. So the company that's behind you is your employer, but they don't treat you as your employer."*

To reduce the risk of brand image damage because of code that is not up to the standards the community expects, one interviewee noted the value of educating developers on the importance of following community specific standards. Two other interviewees said they thoroughly review the code internally

before making it public. Although there is a review process within the community when code is contributed, the inclination to lean on that to find mistakes can lead to damage to the brand. Testing and reviewing the code beforehand reduces the risk:

> *"So I do consider that you need to be careful and try to, you know, be respectful with their standards, take the time and read what they ask and make sure the code is properly indented, that it follows their standards and that it is properly tested and stuff like that."*

Two interviewees noted that holding employees accountable for their actions helps to reduce this risk. Though penalties may not be appropriate for minor issues, having a discussion on improper conduct in open-source communities can help in creating a better understanding of how to behave in these public spaces.

On the policy side, brand risk can also occur when a company contributes to a dubious project, as three interviewees spoke about. If a project, or the leadership of a project, becomes controversial – whether that is due to the type of software, certain business ties or unacceptable statements online – it can reflect poorly on the companies that contribute to that project. Sponsorships or other monetary contributions are particularly susceptible to this risk, as the name and logo of sponsors are usually visible on the website of the project. This makes the connection from a company to the project immediately clear, even to those who do not dive into the code contribution history of the project.

Two interviewees found vetting projects before contributing is a good way to mitigate this risk. This process should include analysis of the quality standards of the project, but also a look into the project owners and important developers in the community. This is not a one-and-done process either. Potential issues with persons in the community can emerge over time. It may be worth it to follow the project, project owners and important contributors on their social media platforms so the company does not get surprised if someone involved starts behaving in an unacceptable way. As one interviewee explained about the process of their company when choosing a project to sponsor or contribute to:

> *"We are very fond of standards: code coverage, quality control, stuff like that, and projects we sponsor need to comply to that as well. (...) We also look at: who is the person or the group behind this [project]? How are they on social media? (...) For each project we look at: what is this project? How does it work? What kind of people are involved? Which people are associated with it?"*

This risk may even occur if an employee contributes to such a project on their own time:

> *"So the risk for us in terms of brand is that if someone went and contributed to software that is harming people, then yes, we do have a problem."*

However, it may not be a viable solution to restrict the projects an employee can contribute to. As another interviewee said:

> *"So there is the people component and I think that's always going to be there because you can't control what people do and it's also: we want people to have the freedom to be themselves too."*

A strategy to reduce the risk of employees contributing to controversial projects on their own time is not immediately clear. It may be a solution to incorporate education on vetting projects for these types of issues into the communication education mentioned before.

Another risk to the image of a company is not contributing back to projects (this topic is discussed in more detail in Chapter 4.12). A company that uses a project and keeps all their development for it private may be frowned upon by the community. This is especially true if the company makes money off the project:

> *"If you're monetising from an open source project you are expected to contribute back. (...) If you continue that pattern [of not contributing] eventually the community might start frowning upon it, they may start being less responsive to you, so if you ever need help with something or you ever need something upstream or you want to contribute, they might start being less responsive."*

Although the risks mentioned in this chapter can influence public opinion of the company, the consequences are strongest in the open-source communities themselves. As the interviewee spoke about in the quote above, the community may be less helpful to the company, which may endanger the benefits open-source offers. Additionally, potential employees may think a company is less attractive if it is controversial in the open-source community:

> *"The brand damage is more towards finding new employees"*

Six interviewees went a step further, they found that contributing to open-source projects was almost always a benefit to brand image:

> *"I think from my experience, my perspective, generally contributing back to open source only ever improves a brand image. Especially if you're a tech company contributing back to open source projects. "*

They believe the fact that the company is active in open-source and wants to contribute is appreciated enough to offset any mistakes that are made. This is good marketing towards business customers, which will be discussed in more detail in Section 4.11. It is also appreciated by employees. According to six interviewees, developers like contributing because they can improve their skills, increase their visibility and marketability because of contributions, and because open-source projects are often fun to work on. This can also be used to attract new talent.

"Contributing to open source contributes immensely to the quality and the knowledge and in general it contributes a lot to the team. The team becomes better in development because they experience this in open source."

Figure 1 shows a visualisation of the causes of brand image risks and the associated risk mitigation strategies.

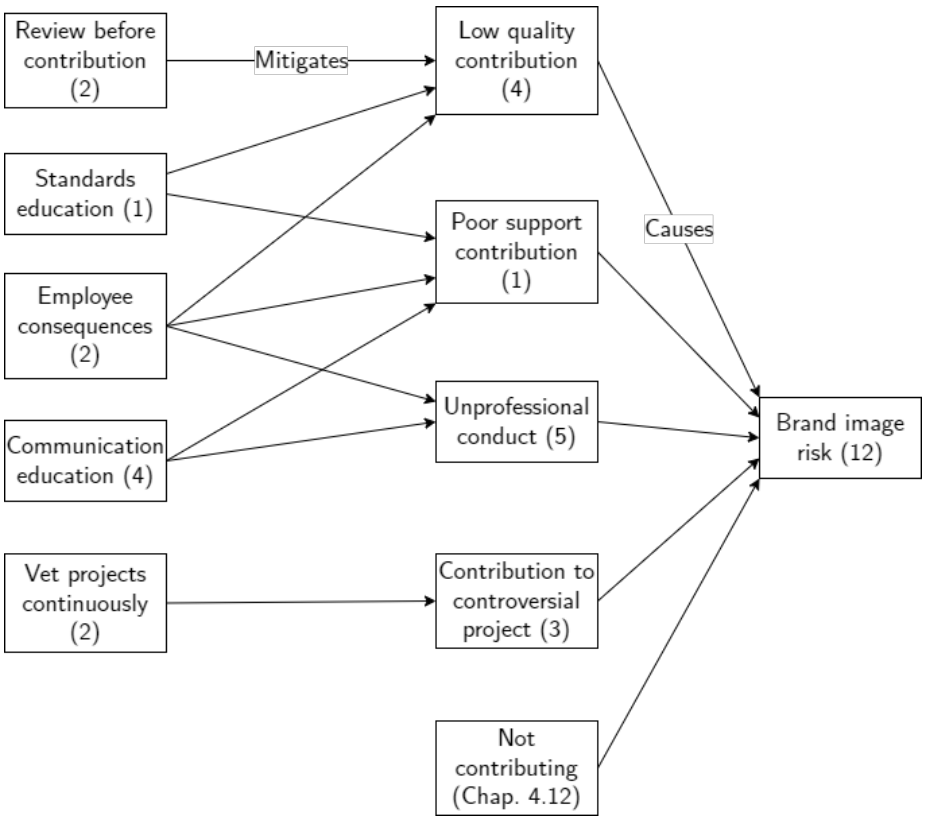

Figure 1: Mitigation strategies for the causes of brand image risks. In parentheses: number of interviewees that encountered the risk or mitigation strategy.

4.5 Rejected Contributions

With every contribution, there is a risk that it is not accepted. The choice to reject a contribution lies with the core developers of a project. Although a contribution may be rejected for technical reasons, this is trivial to solve by fixing the issues and resubmitting. This usually leads to acceptance of the contribution. In the remainder of this chapter, total rejection without option to resubmit will be discussed.

All interviewees agreed that there is a risk that contributions are not accepted, but not all of them agreed on the severity of that risk. Rejection can have multiple causes.

First, it is important to consider the size of the contribution according to three interviewees. Small contributions take less time to review and have a lower impact on the project, so those are easier for maintainers to consider and thus more likely to be accepted. It may then be rewarding to break up larger features into smaller ones that can be proposed one at a time. Over time, the company gains more influence and expertise in the project, larger features become more likely to be accepted.

> *"If I just went in and said: I want to rewrite this project's complete user interface, that is a big gamble for that project. But if we've been contributing for years and we come along and we're a known quantity, we can back up the words we're saying, someone will say: "Alright if you want to contribute the money to rewrite our user interface, we'll let you and we will work with you." So I think that it is always worth it to continue to build those relationships and make the effort to work with those upstreams."*

A feature can also be rejected because the community does not find it useful. This may be the case for features that do not have a use case that is broad enough, according to one interviewee.

> *"I can say that mainly the kind of features that will be rejected by the community or the maintainers are: if it is not well designed, and really you are building a feature for your company and not a feature for the project itself."*

Any feature that is accepted by a project will be maintained by the core developers of that project. Because of that, features should be worth the time and effort to review and maintain, otherwise they will be rejected according to five interviewees.

The other reason for rejection is that the proposed feature is not in line with the direction the maintainers want the project to go in. Features may be rejected more often if communication between the developer and the maintainers is poor. Four interviewees who are active as open-source maintainers spoke about the struggle of receiving pull requests without prior communication. This leads to wasted time on the side of the developer and frustration on the end of the maintainers.

> *"There's been a lot of cases where somebody shows up and proposes something and then they take the time to learn how to implement the feature, without waiting to find out whether we liked the feature or not. And then the contribution just sits there, because we say: "Well, we didn't want that actually.""*

To mitigate the risk of rejected contributions, there are a few steps a company can take. Ten interviewees found that good communication with the community and maintainers is important. Discussion on potential contributions should be had before development is started, and should be continued throughout the development and contribution process.

> *"One of the best ways is kind of talk about what you're planning to do before you do any work. It's a lot quicker to spend half an hour writing out a proposal or a kind of very detailed GitHub issue saying: "This is we want and this is how we plan to implement it." Rather than spending a week building out and then to get rejected. Because then you can have those conversations upfront rather than wasting all that time."*

As the interviewee noted, creating a proposal is much less time intensive than developing a feature first. This proposal should explain the intent of feature and its use cases. It may be beneficial to add a proof of concept as well.

Additionally, by first proposing implementation, others in the community can voice their support. When additional community members back the proposal, maintainers will more readily accept it. However, if it does seem like the company is the only one that would benefit – in which case the contribution would usually be rejected – discussing the proposal with community members and maintainers presents an opportunity to work out a way the proposal would benefit both. Going through these discussions before starting development will prevent wasted time spend on rebuilding the contribution.

Three interviewees noted that discussions of this kind help understand the needs of the community and the project. This understanding can in turn aid in the process of further development for the project and reduce the risk of misalignment with the project.

After a while, when developers in the company gain expertise in the project, this process of proposing and contributing becomes easier. While developers and the company are unknown to the community when they first start contributing, if they are able to show their commitment and growing expertise to the community, contributions are more likely to be accepted according to three interviewees.

> *"In the beginning it is going to be harder but you are continue to do it and eventually people are going to start trusting you and are going to start seeing you as expert on the field and pull requests and contributions are going to go smoother the more and more you contribute."*

A good way to start is to keep the number of developers who are directly involved with the project low. This way, the maintainers and community will get to know them, while they will be able to gain experience with the project. When a good relationship is built and the developers gain experience with the project, additional people can be introduced.

The next step to decrease this risk is to have a developer apply to become a maintainer. This not always an option, some projects may not accept new maintainers. It is also a decision that should not be taken lightly. Becoming a maintainer means that a developer will have responsibilities in the project that are not directly beneficial to the company. They will be involved in discussion about contributions from others, as well as in the reviewing process. These obligations take time away that could be spend on development or other tasks relevant to the company. On the other hand, having an employee as maintainer does give the company influence in the direction of the project, and six interviewees found that contributions will be more easily accepted.

> *"If you have an employee in your company that is a maintainer in that project you can also reduce that risk. So making sure that you are really embedded in it, that you understand how those decisions are made, and that you get in on that maintainership or stewardship of that project"*

If none of the above solutions work and the feature is definitely necessary, the project can be forked. Eight interviewees saw this as a potential solution. In that case, the company has to maintain the feature itself and merge any new releases from the main project with the forked project. Although this is definitely possible, there are some risks involved. This will be discussed in more detail in Section 4.12. In short, however, forking a project means the feature does not receive support or maintenance from the project, any new feature may conflict with it and developers will need to spend time merging any new releases with the fork.

Figure 2 shows a visualisation of the causes of risks involving rejected contributions and the associated risk mitigation strategies.

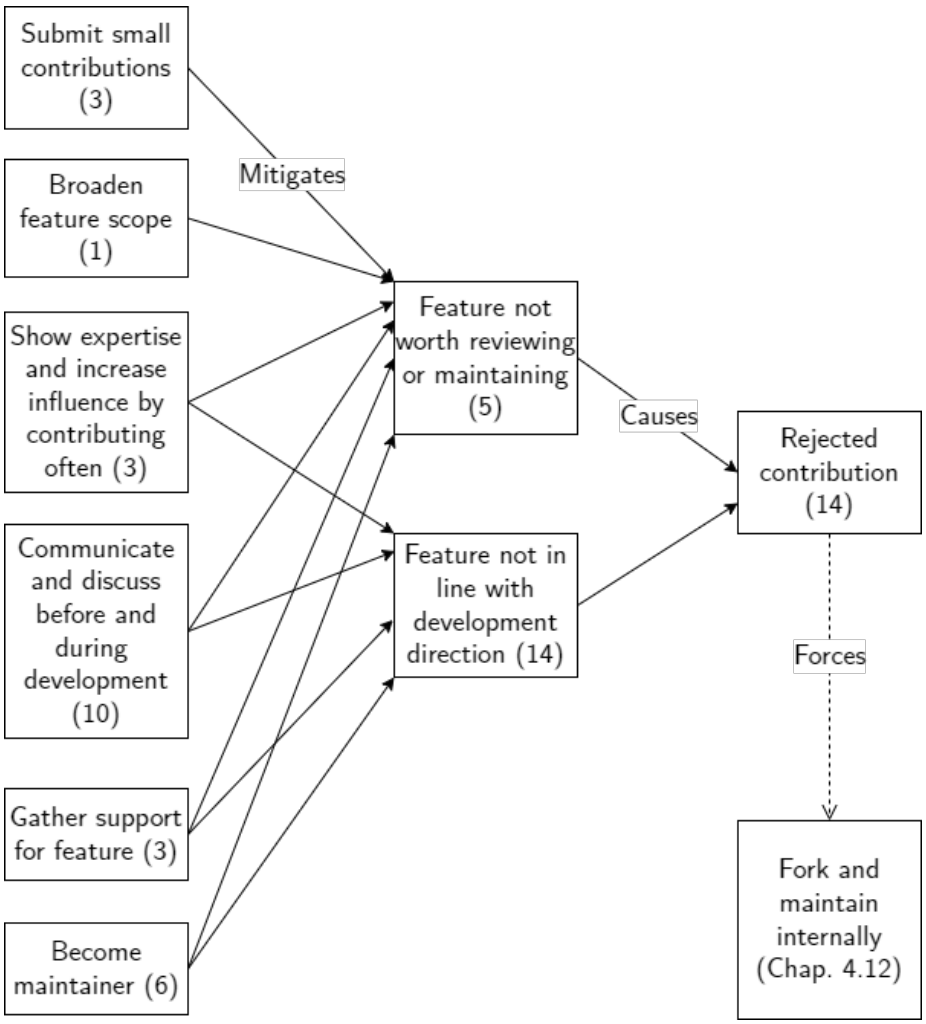

Figure 2: Mitigation strategies for the causes of rejected contribution risks. In parentheses: number of interviewees that encountered the risk or mitigation strategy.

4.6 Financial

Whether there is a financial risk involved in contributing to OSS was a point of contention between interviewees. The discussion came mostly from the question on whether or not there was a risk that contributing would mean investing more development hours. Six interviewees did think so: conforming to the standards set by the community, generalising features, discussing the problem and reworking code that was returned after review were aspects that were mentioned. One interviewee noted that for some of the technical standards it may be possible to use tooling, such as a linter, to reduce the extra work load.

> *"If you're going to open-source something, you know a lot of people will look at it so you just make sure that you are a lot more precise in the code you write; taking care that everything is tested well, etc. Patching bugs as well, that takes about three times as long in open-source I think, that really does take extra time but then you do know that it is correct."*

As discussed in Chapter 4.5, discussing new contributions and gathering support for features is expected when working in open-source, so that takes time as well.

In addition to the extra time that is spent on developing and communicating, it is important to have in mind what the opportunity cost of these hours is. Two interviewees noted that time spent on upstreaming could have been spent on some other feature or issue, so that is a consideration that should be taken into account.

One interviewee noted that while this is indeed an issue in practice, in theory most of these issues should be part of the internal development process as well. In their opinion, these extra costs are not part of contributing to open source, but part of a well working development cycle.

> *"(. . .) only I always refute that part of needing the extra time that is necessary to make your code public, because if there was apparently a need for extra time, then we didn't do it correctly internally."*

In their view, there would be no additional costs involved at all. In the same vein, one other interviewee said that their company has an "upstream first"-strategy. Everything they develop is with open-source in mind. The result of this strategy is that they do not have to rewrite their code if they decide they want to contribute a feature, as it is already generalised, tested and conform to the standards of the open-source project.

Four interviewees found that there is a risk that the contribution process takes a long time. There is no way to predict when a feature will be accepted and when it will be added to a new release. Although this may not cost many man hours, it may mean creating a workaround or an internal fork, so the feature or bug fix can be used within the company without waiting for it to be accepted. This is a hard issue to solve, as it depends entirely on the core developers of the project.

According to one interviewee, there may be infrastructure costs involved, such as testing infrastructure, in some open-source projects. If a company is paying for these costs – either because they began the project or because they starting donating this infrastructure – and they want to decrease this investment without impairing the project, it may be a solution to create more awareness among other companies that these costs exist. This may not be a known fact, and if others are made aware, they may take on some of these costs.

Five interviewees considered the financial gains of contributing to open-source to be greater than the costs. The main benefit four of them mentioned was that everything that is accepted by a project will be maintained by that project. The company itself takes on no costs for that in the future.

Some also noted that the use of open-source software already saves costs that would otherwise have been incurred. For example, even though a company might spend development time to fix a bug in a project that would then be contributed for free, at the same time there are other companies and users in the community who fix others bugs for free that the company benefits from as well. Additionally, the upfront cost of developing or purchasing a software product is saved as well when using open-source. Following this logic, contributing to an open-source project may involve some costs, but the costs saved from using that project at all are much higher.

> *"The extra development time we spend to learn that one extra change, in a way, is us contributing our money and time for the benefit we've gained of all of the hundred, if not thousands hours of development time that has already gone into that project."*

Figure 3 shows a visualisation of the causes of financial risks and the associated risk mitigation strategies.

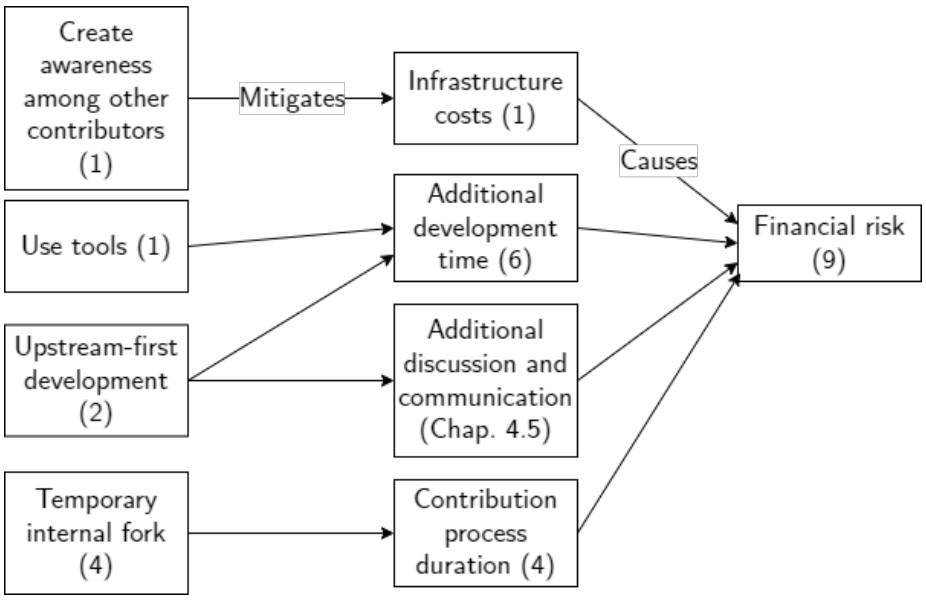

Figure 3: Mitigation strategies for the causes of financial risks. In parentheses: number of interviewees that encountered the risk or mitigation strategy.

4.7 Intellectual Property

Nine interviewees thought there were risks around intellectual property when contributing to open-source software projects. These risks arise in several forms. The first is in the discussion what proprietary code should and should not be shared. When working in open-source software, the balance between sharing or not sharing something than can be contributed is not always easy to find. Usually there are advantages and disadvantages to both, and the best decision may not immediately be clear. The risk then, is that the wrong choice is made. The risks that are involved with not contributing something will be discussed in more detail in 4.12. The risks that come with contributing while it was not the correct choice will be described here.

Three interviewees noted that when proprietary code is shared, the ability to generate direct revenue from it is lost.

> *"The first thing I think that companies face, especially if its an IP driven company, is the risk of proprietary IP that is not ready to be open sourced, or let's say IP that was going to be exploited and monetised, being put out under a license in open source which sort of freely grants those IP rights. And I think that's one of the biggest risks that I think not just [company] but any company out here would face."*

Whether or not this is a risk, depends on the way a company generates revenue. If it is product based company, the risk is much higher than if it is a service based company. One interviewee mentioned that their company sometimes chooses to monetise a feature first, and then contribute it at a later date. This may be a good choice if a middle road has to be found.

Five interviewees found that it is possible to lose competitive advantage by contributing a feature or bug fix that should have been kept internal.

> *"There is a risk there that engineers want to donate engineering time, work on open source, whereas management tends to be more careful about giving away that kind of unique selling points or giving away the stuff that makes that company. And so there is a risk there that engineers will submit stuff that might kind of blur the lines a little bit."*

The same can happen when comments in the contributed code say too much about upcoming features or products, according to three interviewees. If competitors read these, they may have the ability to have an answer to the releases the company has planned. Developers – assuming they interact with the open-source community themselves – could easily make either of these mistakes. Five interviewees found that to reduce this risk, it is useful to have a discussion on what can and cannot be shared before any interaction with a community takes place. Furthermore, one interviewee noted that the risk can be further reduced by requiring contributions to be reviewed internally before they are sent to the project.

> *"Each contribution to open source needs to be registered, that means which project you are contributing, if you are originating a new project as well, you should go to the intellectual property analysis to see if there is something that could be patentable for the company or not."*

While these discussions do reduce the risk of any intellectual property being exposed when it should not, it does make the process of contributing more difficult and time consuming. This is a consideration that should be weighed against these risks.

Another issue is the ownership of the intellectual property. Because different countries have different laws, it is outside the scope of this thesis to go in depth on those issues. However, four interviewees considered it a risk that either employees – or business customers, if a company develops software for others – disagree with the company on who owns the intellectual property. This can be even harder if an employee is hired because of their work on an open-source project, especially if it is a project they have started themselves. If developers work on projects in their free time that are in the same field as the company operates in, issues around ownership of ideas and intellectual property can also arise.

To prevent these problems, two interviewees proposed that companies can include these topics in their contracts with employees. In these clauses, ownership of code developed during work hours and outside of work hours should be included. It should be clear when a developer can contribute without issue and when they should check with the employer, both as an employee and as a volunteer in their own time. These clauses could also include when and to which projects developers should contribute with their work accounts and when they should not. They could also include an avenue for an employee to discuss with the company whether some contribution can be made.

In contracts with business customers, it should be made clear which company owns the code, and under whose name this code will be contributed.

Finally, a company may encounter issues with patent trolls according to an interviewee. These persons or companies claim ownership of a piece of software and take companies to court to extort money from them. According to this interviewee, companies can reduce this risk by defensively patenting their contributions. Another interviewee mentioned this as a good practice in general.

Figure 4 shows a visualisation of the causes of intellectual property risks and the associated risk mitigation strategies. Note that some risk mitigation strategies were mentioned more often than their respective risk. This means some interviewees did discuss the way their company handles an issue, and that they did not see the issue as a risk. This will occur in later chapters as well.

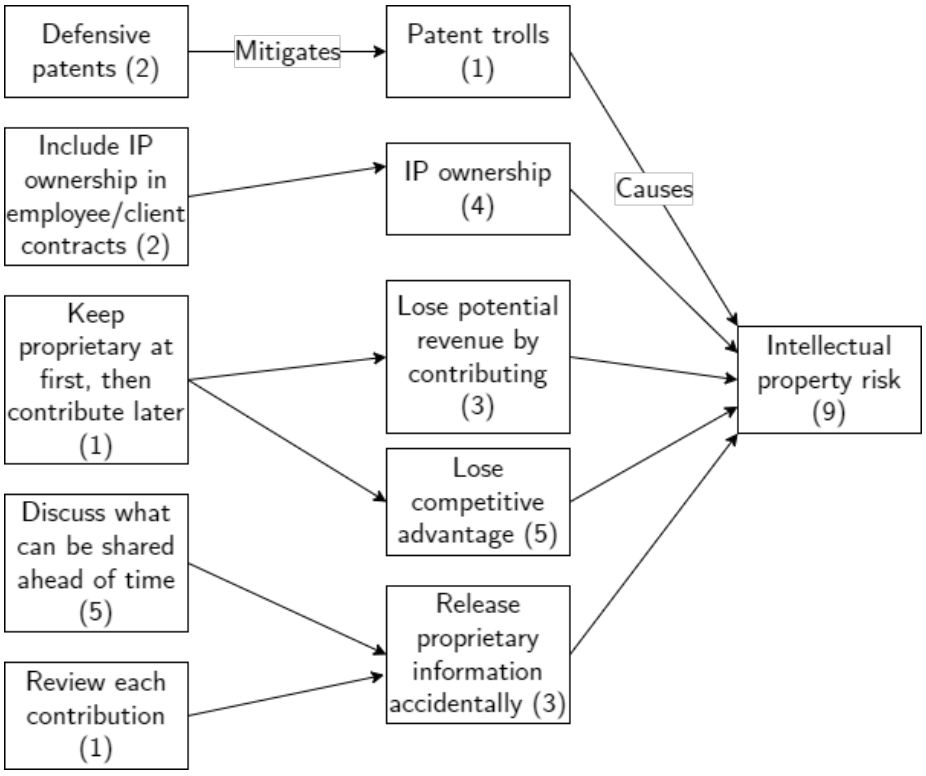

Figure 4: Mitigation strategies for the causes of intellectual property risks. In parentheses: number of interviewees that encountered the risk or mitigation strategy.

4.8 License

Ten interviewees considered licenses to be a risk factor when contributing to open-source projects. There are many licenses and contributor license agreements – additional licenses that are relevant for contributors but not for consumers –, which differ from project to project. It is crucial to understand what these licenses mean and what the consequences of using projects with these licenses are.

As discussed in 2.3, there are (highly) restrictive licenses that make it mandatory to contribute changes to the source code or to open up entire software projects that use the open-source project if it is distributed. Four interviewees mentioned that this is something they see as a risk. If a single developer is not careful in their choice of libraries, they risk having to open up the entire software project.

> *"The real risk is when you allow your engineers to pull in any project or any dependency that might have something like a GPL or that kind of license where you really do risk exposing your companies source code, because those licenses say that if you have a dependency that has that license then that code also has to be open sourced."*

While it may be tempting to ignore this issue, as there is a chance that not conforming to a license is never discovered, the consequences can be dire. This risk is greater if a company not only consumes this software, but also contributes to it, as it is to be expected that they use it in one of their own projects. Especially for larger corporations it may be harder to get away with. One interviewee mentioned that because of the high downside, investors do really want companies to comply. As such, there is a risk – besides the risk of having to make all code public – that financial opportunities are lost if license terms are not followed.

Permissive licenses are easier to work with, they require very little of developers that use projects under their terms. However, anyone else can use the source code contributed to these projects in any way they wish, which three interviewees saw as a potential risk. Anything that is contributed to it can be reused in proprietary software of others.

> *"Apache and MIT, all awesome, but also permissive, which means if I put something out someone can fork it and do whatever they want to do with it."*

Projects that do not have a license attached to them also form a risk, according to one interviewee. In that case it is unclear who owns what code and how the software can be used. They recommend not using those projects.

Seven interviewees found that to mitigate the risks around licenses, it is important that the company is aware of what types of licenses exist and what they entail. Developers who work with open-source software should be aware of the different licenses as well. Four of those interviewees said they work with

legal experts to vet licenses. They can also support developers when they need information on licenses.

Six interviewees noted that it is possible to choose specific licenses that developers are allowed to use ahead of time. This can be defined either as a company-wide policy, or for each project separately. In exchange for flexibility, the company makes sure that projects with licenses that pose a risk are not used. To support this, package managers can be used that enforce these rules.

When working on a project where open-source projects with (highly) restrictive licenses are used, extra care should be taken to mitigate potential risks. One interviewee argued that clear separation between proprietary code and open-source code should be enforced. This ensures no open-source code under such a license contaminates the proprietary code. In addition, if two developers work on similar projects, one of which is open-source and one of which is proprietary, there is a risk of cross-contamination which can be reduced by making sure they do not work together. Finally, one interviewee found that reviewing all code that is contributed can reduce the risk. A review is then not only focused on code quality requirements, but also on adherence to the license.

Figure 5 shows a visualisation of the causes of license risks and the associated risk mitigation strategies.

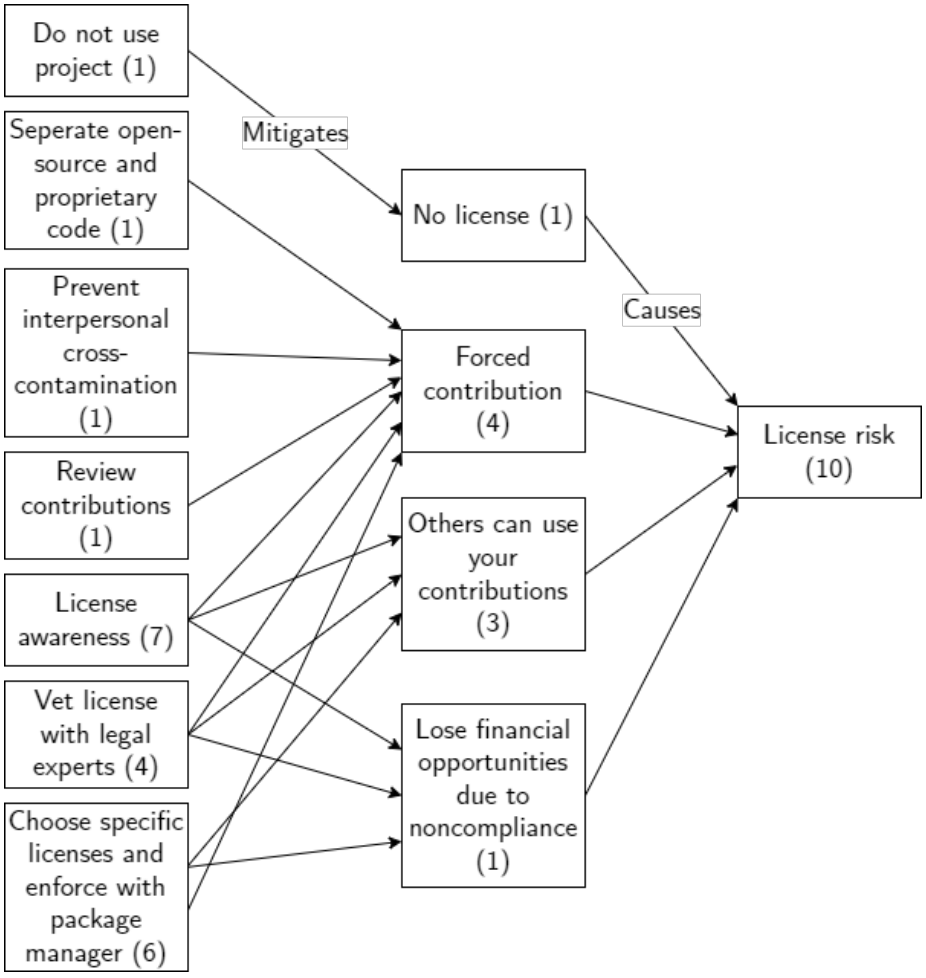

Figure 5: Mitigation strategies for the causes of license risks. In parentheses: number of interviewees that encountered the risk or mitigation strategy.

4.9 Security

Five interviewees thought there was no more risk of introducing security issues into the code than in proprietary software. Two even found that open-source projects carry less risk of introducing security vulnerabilities. Because contributed code is reviewed by maintainers of the project, there is a bigger chance that security issues in it are caught according to them.

This does mean however, that there still is a chance of introducing security flaws. Seven interviewees saw this as a possibility, but as noted before they found that the impact of this is similar to proprietary software in terms negative consequences directly related to the issue. However, there are some differences in risk regarding indirect consequences. For both open-source and software that is sold, security vulnerabilities can lead to negative impact on the brand image, as discussed in Chapter 4.4. For internal software this risk is lower – unless customers or other stakeholders are greatly affected by it. This risk may be larger for security vulnerabilities that are introduced in open-source projects, as even issues that are not exploited can be found by other parties. This could lead to a negative impact on the brand image of the company that contributed the part of the software that exposed a security issue.

> *"As a company, you really need to know what you are doing. If you are the one that introduces a security bug, and of course that has happened, that has happened multiple times, then that hurts."*

Five interviewees found that another potential security risk is the exposure of information, login credentials, private keys or customer information. As discussed in Chapter 4.7, leaking proprietary information in contributions can happen. While the mitigation steps explained there are applicable in this case as well, exposing this type of information can have a higher impact than exposing some intellectual property. Because of that, more rigorous prevention may be necessary.

According to four interviewees, the way of reducing these risks is implementing a review process before anything is contributed. Whether it is proofreading a proposal or an in-depth code review for a feature that will be contributed, another pair of eyes will catch mistakes that the creator might not have found.

> *"You can make sure that your pull requests are reviewed by people who have knowledge of the system, knowledge of those open source projects and that they have security knowledge."*

The risk can also be reduced by offering security training to developers according to two interviewees. Awareness about potential issues will help developers prevent mistakes proactively. Two interviewees also discussed that the attitude of developers towards potential security issues could either be a risk factor, or something that reduces the risk. There are also security tools and scanners that can be used to mitigate the risk, as noted by seven interviewees. Using these tools as support for employees during the development and review processes will help them catch potential issues.

Usually, the project has a review process for contributions as well. Before any code is added to the production version of the software, maintainers will consider potential issues. According to six interviewees, this is an additional barrier against security vulnerabilities. Finding a healthy project with a capable team is thus important, according to one interviewee. It does not help with leaked information, as that will be out there from the moment a pull request is created.

Figure 6 shows a visualisation of the causes of security risks and the associated risk mitigation strategies.

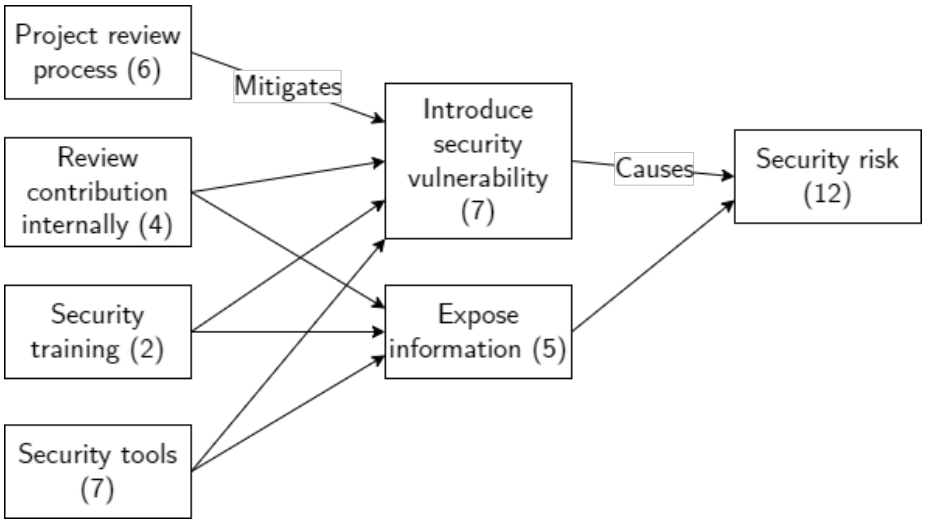

Figure 6: Mitigation strategies for the causes of security risks. In parentheses: number of interviewees that encountered the risk or mitigation strategy.

4.10 Sustainability

Four interviewees brought up that, as with any software, there is a risk with open-source projects that they will cease to be supported at some point. As three of them noted, this is a larger risk in smaller projects. If a project has a small amount of maintainers, it may cease development if they lose one or multiple of them. The risk is even greater for projects that are maintained by a single person. However, if the project has backing by one or more companies, it can be a signal that it is healthy and will exist for longer, according to one interviewee. They also noted that projects that were accommodated by large cloud vendors were more likely to continue development as well.

Many widely used open-source software projects are ran volunteers on their own time. Some of those volunteer developers spend a lot of time on those projects, without much return. One interviewee feared that these types of projects may fail because larger corporations use these projects without compensating the developers. While it may be feasible for some to continue developing these projects, it is not always sustainable. This interviewee, along with one other, found the easiest way to reduce this risk is to offer monetary support to these maintainers so it is financially viable for them to continue with the project These donations can either be a one-time donation or continuous support.

Two interviewees found another sustainability risk in the fact that employees can change roles or move to another company. This can be a risk on both the consuming and contributing side. If the employee of a company works on an open-source project as part of their job, it is to be expected that they will stop that work when they leave. If that happens, a project – especially a smaller one – suddenly loses a contributor and their knowledge, which can negatively impact future development. This can negatively impact the reputation of the company and hamper continued cooperation if it is not handled well. Introducing another employee to the project before the current employee moves to another role or company can mitigate this, according to one interviewee. Another possibility is to allow the employee to contribute with a personal account instead of one linked to the company if they want. In that case, they may be able to continue development, even in their new role according to one interviewee. Finally, the choice can be made not to introduce an employee to the project at all and instead supporting the project financially.

> *"Maybe the more prudent route would be not dedicating an FTE, but take that money, give it as a grant, which has a structured scope, one year contract for the project that is guaranteed because it is not connected to the employment or the role of any individual."*

On the other side of the equation, if a company uses a project that has few maintainers, some of which maintain it as part of their job, there is a risk as well. If one of those maintainers, or the single maintainer, leave their company, the project they rely upon may cease development. Two interviewees fount that to mitigate this risk, a company should choose projects that have multiple active maintainers.

One interviewee noted that if a project does stop being developed, a company can choose to fork the project and continue it themselves. This is an advantage over proprietary software, where a company that uses it has no choice but to switch to some other software.

Figure 7 shows a visualisation of the causes of sustainability risks and the associated risk mitigation strategies.

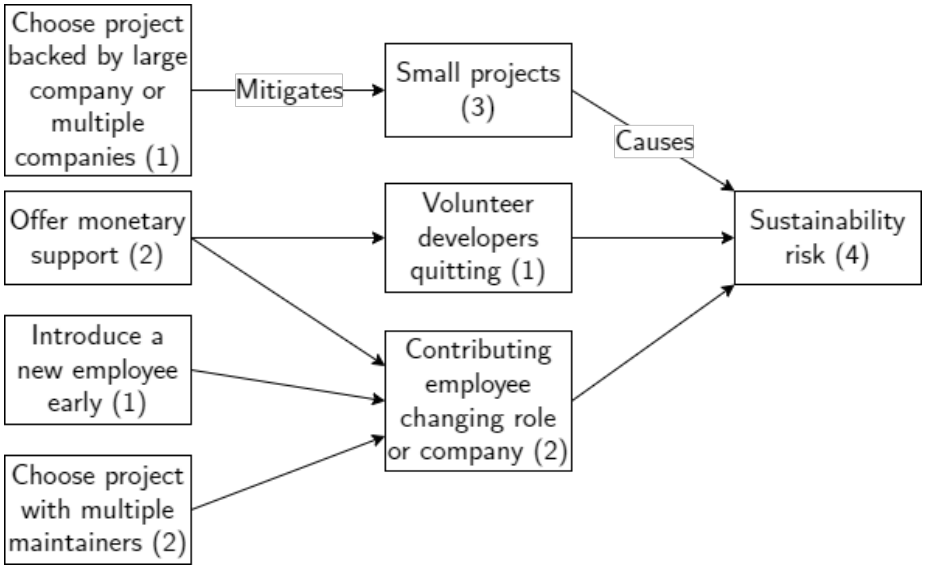

Figure 7: Mitigation strategies for the causes of sustainability risks. In parentheses: number of interviewees that encountered the risk or mitigation strategy.

4.11 Business Customers

As shown in table 1, multiple interviewees are part of software consultancy companies. Those interviewees mentioned risks that are involved in that part of the software development business. Although not all companies will encounter these risks, they will be discussed here nonetheless.

Two interviewees considered the contracts and non-disclosure agreements (NDA) that are usually signed with business customers as a potential risk factor. Issues can arise when developers are not made aware of exactly what they can and cannot share in open-source communities. They should also be aware of this when developing for their own company. However, with a third party involved potential legal disputes and fees increase the severity of this risk.

> *"So the main risk I see, as an employee from my current position, is about contract terms and NDAs. Where one side wants one thing, another side commits to that thing and then individual employees are starting without knowing these details about the agreements to be proactive. It is open source, (...) everyone can see it, and such activity can have opposite consequences of what the contract terms where about initially."*

It is important to consider the expectations around contribution from both companies. For the developing company, it is generally beneficial to contribute, as it reduces certain risks (as will be discussed in Section 4.12). Additionally, contributing to open-source projects is good for business to business marketing, according to seven interviewees. On the other hand, the customer company may want to keep some or all of the software as proprietary code so their competitors do not receive the benefits that they have just paid for. It is also possible they want to delay contributing to open-source projects so their competitors do not know what they are working on.

> *"For the software on the custom things you develop for a customer, so especially as a service company, the customer decides whether that can be opened up or not."*

Although only two interviewees saw contract issues as a risk, four interviewees noted that it was important for the two companies to come to a clear agreement on what can and cannot be contributed. They should then stay in touch so they can clarify if it is unclear at any point during development. Part of this discussion may involve convincing the customer that contributing to open-source benefits them as well. Explaining all the benefits and risks involved with contributing – or not contributing – may convince the customer that opening (part of) the software is indeed the right choice.

> *"The customer may think [contributing is risky], but open-source does have a lot of benefits, you don't have to maintain it yourself for example, so why wouldn't you want to give back? So we do always try to convey our point of view to the customer."*

One interviewee noted that even if the customer did not mind that something would be contributed, they might not want to pay for the time it takes to do so.

> *"So you know we encounter a bug in the open source project, "so yeah can you guys fix that for us?, of course we fix it, we get paid for that, but "hey do you mind if I contribute back to the project?" and they say: "sure you can do that but not on our time". So that is one big risk: customers do not want to pay for us to go back and contribute to the open source projects."*

If this is an issue for a company it can – if it has the option – choose to only work with companies that do pay for the time that is spend on contributing, according to one interviewee. This interviewee found that another way to deal with this issue is to offer a discount to customers if that means the code can be contributed. One other concurred, though they saw this as a normal part of the discussion with clients, not as a risk factor. Offering a discount is a trade-off that can be worth it to build a portfolio for the company and to position themselves as experts, as it can lead to more customers down the road.

> *"For example we don't charge the full price for the project, we absorb some of the costs, but on the other hand we can use that time and push it to the project. And we absorb that cost, but on the other end we position ourselves as experts on the project so that eventually can bring us more customers."*

One interviewee said they deal with the issue by just not fixing a bug if the customer does not want to pay for it and it is not breaking anything important. Finally, one interviewee noted that they think the hours spend on fixing bugs that were an issue for a customer should be invoiced either way as it is not really important whether the bug is in the proprietary code of the company or in the open-source code.

> *"I mean, if I solve a bug in their codebase it is for sure invoiceable hours, but solving a bug in third party software which was a problem for a customer should still be invoiceable I think."*

One interviewee noted that this discussion may not always be an issue. Companies may not care about the small details in the development process, as long as certain deadlines for the entire project are met. In that case, the choice to spend time on contributing small fixes and supporting tools may be left to the project manager and not the customer.

> *"The client pays for some big roadmap item delivered and usually does not care about small details."*

There are some benefits that are specific to software consultancy or support companies. One of those benefits is building a brand as an expert in the field,

as discussed in Chapter 4.4 as well. This can lead to other companies that use the open-source software to consider the company if they need some bug fixed or feature developed.

> *"(...) contributing back to the open source project makes you look like an expert and is an indirect way of marketing yourself as an expert, and eventually we have had customers come to us and say: "Hey, we noticed you developed this part, we would like some help, are you open for work?"."*

Three interviewees found the same to be true for support contracts as well. Open-source projects generally do not have a support system that proprietary software packages do have, as there is not always a single owner. The community around it is usually happy to help, but because they do it for free this support can be inconsistent and varying in quality. However, companies often want some way of receiving the support they need. Because of that, there is room for companies that understand the project to fill that demand.

> *"Customers come to us because they want support, they want to be able to pick up a phone and say: "we have a problem, can you help?". So for us it is not the intellectual property that matters, it is the ability to answer that phone call and have that intelligence within developers and that experience and knowledge."*

Figure 8 shows a visualisation of the causes of business customer risks and the associated risk mitigation strategies.

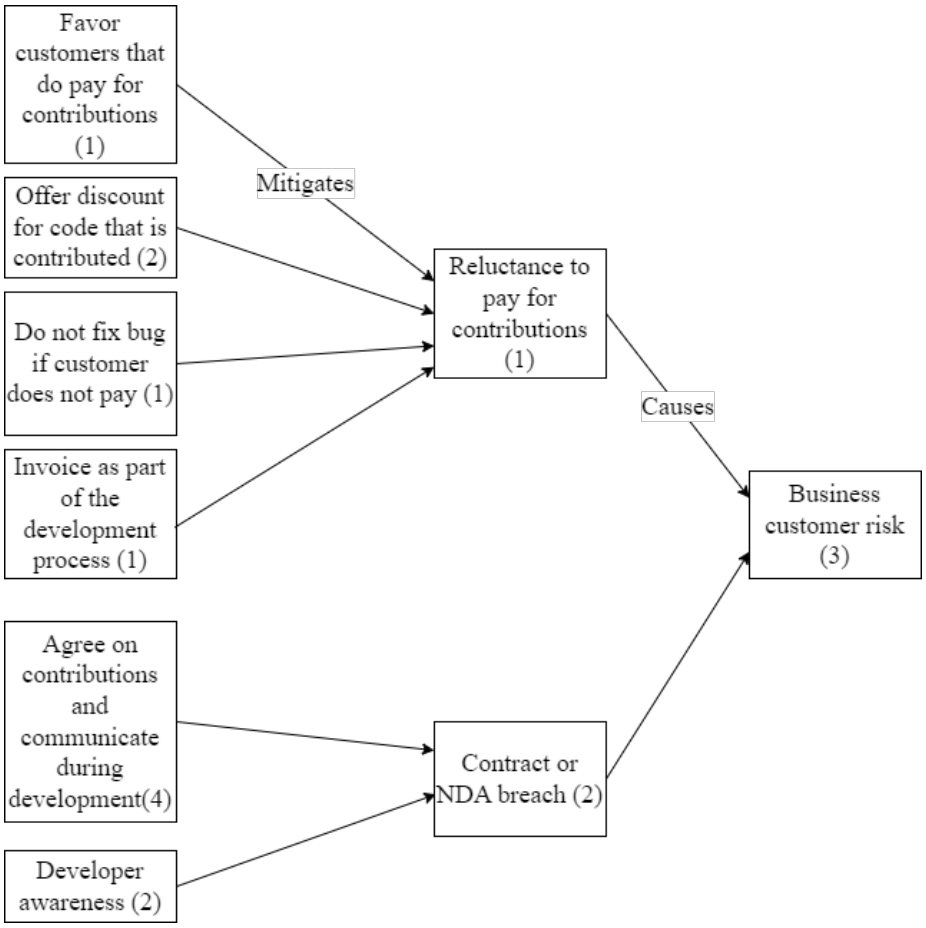

Figure 8: Mitigation strategies for the causes of business customer risks. In parentheses: number of interviewees that encountered the risk or mitigation strategy.

4.12 Not Contributing

Although all risks that were discussed in previous chapters emerged when contributing to open-source, there are also risks that come up when the decision is made not to contribute. While it is perfectly valid – and not inherently risky – to only consume open-source software, this chapter considers the case where a company does develop code for a project and then decides not to contribute it. These risks also arise when software cannot be contributed due to rejection by the project maintainers, as discussed in Chapter 4.5. Then, the company can choose to either patch the feature – if that is possible with that particular project – or to maintain an internal fork of the project.

One risk of not contributing software that has been developed to an open-source project, is the fact that the company misses out on the benefits of open-source. First, two interviewees noted that the software is not reviewed by members of the community. This means missing out on their expert view on the code, with potentially lower quality or missed issues as a result.

Another loss, one that nine interviewees mentioned, is the fact that features that are not contributed are not maintained by the community. This is a cost that can be avoided by contributing the code. The risk here is that code that is maintained by the company may become incompatible with new releases of the open-source project. At that point, even more time has to be spend to realign the feature with the main project, or the software cannot be updated. Especially when the project releases security updates, there may be a need to do this abruptly. Three interviewees saw that as a potential risk.

> *"If you patch the project or fork the project, you're going to be out of date, you're not going to have security updates. Updating is going to be really hard, because if you're patching you are going to need to figure out how to merge things yourself and I don't think that's a good use of time."*

Seven interviewees found that with time, this technical debt increases. A feature that fits well into a project right now, may diverge from the direction of the project down the line. If a company is dependent on that feature, they will need to continuously put time and effort into that feature to keep it up to date. This may be doable for one or a few features on a single project, but the scope of this issue will grow as more features for more projects are kept private.

> *"I know from experience from a previous company I worked for: the moment you [create a fork and maintain it yourself] for one project, it's okay. However, if all your dependencies are forked stuff, then it will cost a you a lot of time and a lot of money to maintain that."*

One interviewee found that the cost of integrating a fork with each update can become so vast, that it is cheaper and easier to split off from the project entirely. Then the company is responsible for the maintenance and upkeep of the entire fork. Consequently, the company can then no longer extract value from the open-source project itself.

One way to mitigate the risk of rising time investment and costs to maintain a fork – and to reduce the risk of having to split the fork from the main project – is to keep the tech debt as low as possible. This is not a trivial task. One element of this process is to update the fork regularly and to keep it as close to the main project as possible. The less the two diverge, the less time and effort it takes to keep the fork up to date. It is important to support the developers who do this by giving them the time to work on this, according to three interviewees. One interviewee added that custom features should be documented and that the documentation should be kept up to date with new development.

One way to get out in front of this, is to choose projects that use a plugin architecture, as two interviewees noted. For those projects, it is easier to maintain features that are not contributed, as they can be included as a plugin. This reduces the amount of work it takes to maintain the feature, as there is less risk of conflicts and complicated merging. However, this will not be possible for all types of software and it should only be one part of the selection of an open-source project.

One interviewee mentioned that another way to reduce this risk is to build a feature that will not be contributed into the proprietary software of the company, if that is possible. In that case, the company will not have to maintain a separate fork for the project.

As five interviewees noted, it may be worth reconsidering whether it is worth it to keep code internal. The decision to not contribute something is easily made, it is cheaper, it is straightforward and it carries little immediate risk. However, there are perpetual long term costs involved in maintaining and merging the fork or patch. Reconsidering whether the decision was not made prematurely and comparing the costs and risks involved in contributing versus not contributing can mitigate the risk of making the wrong choice.

> *"I think you do sit down and start thinking what to do: what is the cost of maintaining my fork? Versus what is the benefit of maintaining that fork? (...) And if the benefits of the fork outweigh the costs, you just kind of suck it up, right? But in general I think consistently trying to get things upstream as much as possible is the best risk mitigation strategy."*

As one interviewee noted, there may be an inclination to add features or change something just because it is possible. If the decision is made that a feature will not be contributed, the company should consider whether it is worth it to develop and maintain it at all. Just because it can be done, does not mean that it should be done, as maintaining a separate fork or patch can be costly, as discussed before.

> *"But you don't necessarily have to change something, maybe it is also okay to reduce risk to change as a company towards what those open source programs can actually do, instead of to say: "we will change it because we can". I think many companies, and then I do*

speak from some experience, let themselves go into "it is changeable so will we do that"."

If a company is active in the open-source community, but never or rarely contributes, there may be even more negative consequences. Two interviewees found that there is a risk that the community will respond less actively to requests for support. If they do not receive contributions in return for the time they spend helping the company, they will be less willing to help in the future. This may lead to less understanding of the project and its development direction for the developers, which increases the difficulty of developing for the project. Additionally, it may be bad for the brand image of the company as discussed in Chapter 4.4. Especially other people in the open-source community – including potential employees – may have issues with non-contributing companies.

> *"I think it's kind of expected you know? If you're monetising from an open source project you are expected to contribute back. Because it is not illegal to make money out of an open source project, there are conditions in that you can even extend that open source project and sell that extension. There are licenses that allow that. But if you continue that pattern eventually the community might start frowning upon it."*

By not contributing a company also does not interact as much with a community. They do not show their expertise in the project and they do not build relationships with the maintainers. Consequently, it will be harder for them to build influence in the project according to three interviewees.

> *"If you really rely on something and you don't contribute there's always going to be a risk: that it goes in a direction you don't want it to, or that it becomes incompatible, or that it goes away, all of that can happen."*

No direct ways to mitigate these specific risks was found. The only possible solution would be to reconsider contributing, as stated above.

Figure 9 shows a visualisation of the causes of the risks of not contributing and the associated risk mitigation strategies.

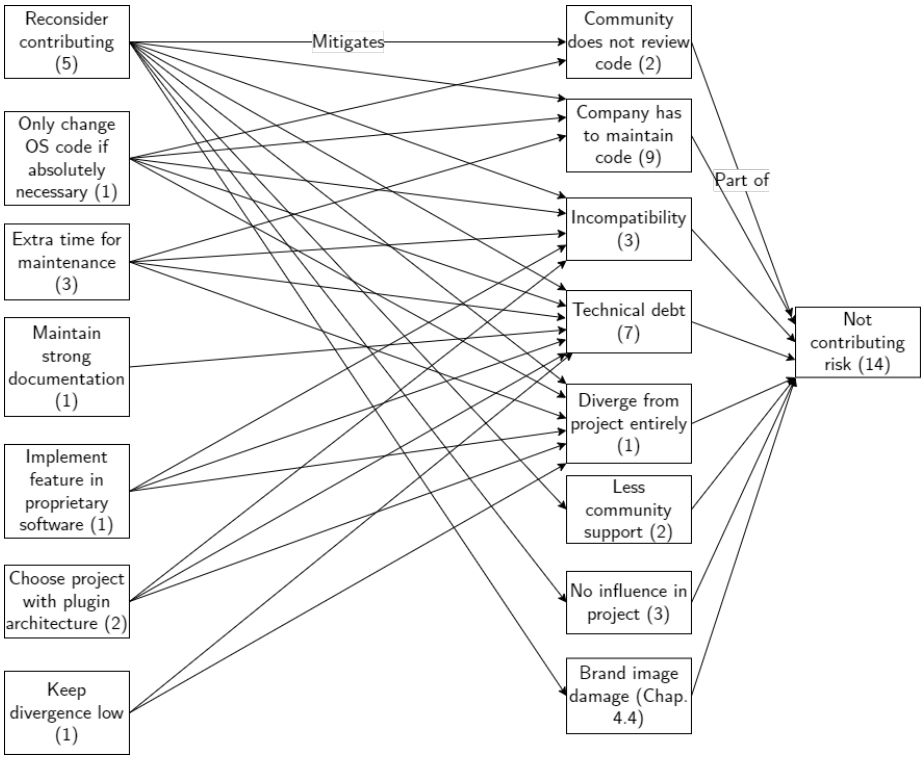

Figure 9: Mitigation strategies for the causes ofthe risks of not contributing. In parentheses: number of interviewees that encountered the risk or mitigation strategy.

4.13 General Mitigation

Besides the specific risk mitigation strategies explained in the previous chapters, there are some actions that can be taken to reduce open-source risk in general.

Educating employees on how to use and properly contribute to open-source software is vital according to four interviewees. Knowledgeable software developers will make less mistakes when interacting with and contributing to open-source software. Understanding how open-source works and what risks are involved helps employees work on it with confidence.

As seven interviewees noted, communication is another factor in reducing the risk around open-source. Expectations management is the first part of this, both within the company and between the company and open-source projects. Communication with the open source projects that will be contributed to will prevent unnecessary discussions down the line. Clearly stating what will be contributed and whether help is needed in doing so will manage expectations on the side of the project. Asking for feedback ahead of time on planned contributions will manage expectations on the company side.

> *"The other thing would be to talk to the maintainers of the project and find out if there is any specifics to be aware of. Especially for technical and perhaps legal things, talking to the specific maintainers is probably a good idea."*

One interviewee mentioned that this communication will be easier if employees work as a developer on a certain project full-time. Embedding an employee in a project this way creates a strong connection between company and project.

Within the company, there should be clear communication on the specific goals and expectations for contributing. Whether contributing is a goal in and of itself, or whether it is just a means to an end should be discussed or instructed clearly. It should be clear what can and cannot be contributed and what can and cannot be discussed in communities. Whether both bug fixes and features can be contributed should be decided as well. Interviewees did not agree on whether there was a different level of risk for contributing features or contributing bug fixes. Some interviewees found contributing bug fixes to carry less risk, as those are often smaller contributions. However, other interviewees found that it was just as risky as contributing features.

One interviewee found that a dedicated open-source program office can be a valuable factor in enabling this communication. Not only can they function as a link between employees working on open-source and as a source of information when employees need information, they can also serve as advocates for open-source within the company. Additionally they can be an important factor in license compliance and community management.

If there is no expertise on open-source software within the company, it can be a good idea it to consider hiring advisers to inform the company on the choices that should be made according to five interviewees. Outside experts can help set up an open-source program office, create guidelines and teach employees what they need to know before starting to contribute to open-source.

As nine interviewees argued, formalising a policy or guidelines for the open-source strategy – including risk mitigation measures – is important for the company. With this policy, it will be easier for employees to understand what they can and cannot do, and what they have to do to reduce risks. This policy can also include smaller points of discussions, such as which accounts – personal or company-owned – should be used when contributing, and how much and what due diligence should be performed before a project is contributed to. The policy should also readily be available for employees to check whether they are following them correctly. Additionally, other decisions that were made and things that employees learned while working with open-source should be documented for later reference as well.

> *"I'd say that clear policies are the best way. So encouragement from senior management that contributing to open source is good helps enable those things. Otherwise people are generally a bit worried about contributing because they are not sure about the process and if it's allowed."*

Six interviewees said that policy can be supported by tooling. It should be considered what tools are available and how strict the tools should be. If stricter tools are used, it will become harder to contribute and the process will become less flexible. So while strict tools do reduce risk the most, it will come at a cost. It should therefore be balanced around risks that occur often or are high impact. Two interviewees also mentioned that employees should also be trusted to follow the policy without enforcing it entirely with tools.

Figure 10 shows a visualisation general risk mitigation strategies.

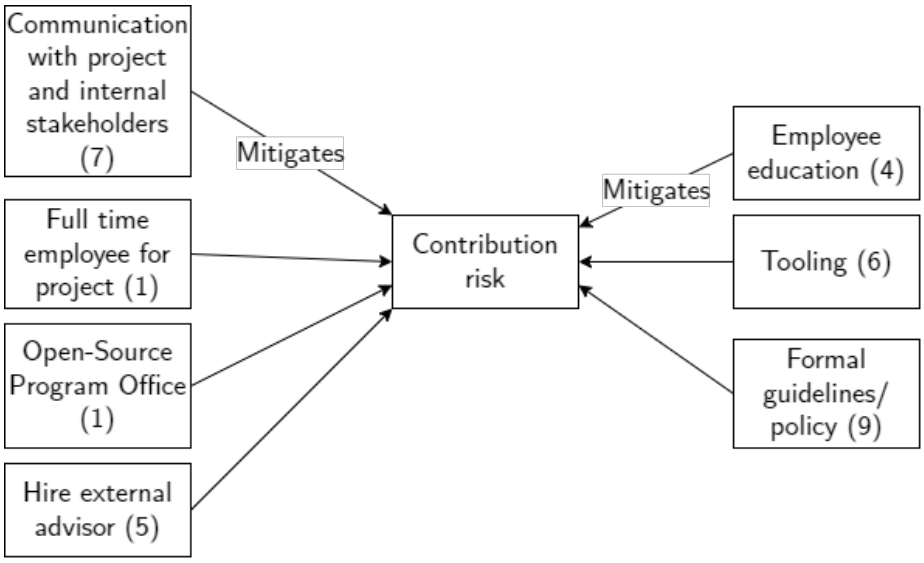

Figure 10: General risk mitigation strategies. In parentheses: number of interviewees that encountered the risk mitigation strategy.

5 Discussion

5.1 Interpreting the Results

Not Contributing

It is notable that all interviewees spoke about the risks of not contributing. While it was expected that some would be against this practice for ethical reasons, the business related reasons predominated. The main issue that was mentioned was the cost of maintaining a patch or fork. This was found by Butler et al. (2019) and Linåker and Regnell (2020) as well. Not contributing also means losing out on some of the benefits that open-source software offers. The company has to maintain the code itself, receives no code review from the experts on the project and it creates an ever-growing technical debt. Each new release of the open-source software forces the company to spend time updating its fork or patch to avoid incompatibility, or it has to choose to diverge from the project entirely. While this may not be an issue for a few important and valuable changes, the more changes for which this is the case, the bigger the problem grows. This becomes especially bothersome if the reasoning behind not contributing, or even developing the the feature in the first place, becomes less clear or less important over time. Keeping a fork or patch private for a longer period could make it impossible to find a path back towards merging with the main project. In those cases the choice to diverge entirely can be made, but then the entire task of maintaining the fork falls on the company. This can incur enormous costs over time, which outweigh the initial benefits of keeping the code private. For this reason, it is important to reconsider contributing a change to the code, even if it seems to be too valuable to contribute. Only when the benefits really outweigh the cost should the code be kept private. Even then, if the feature becomes less valuable or the cost of integrating it with the main code of the project becomes too cumbersome, it can be valuable to contribute it at a later point.

Rejected Contributions

Related to the choice of not contributing code is the risk that contributions are rejected. When one does want to contribute and receives a rejection, the cost of maintaining a fork will be larger than the benefits. That makes this risk important to mitigate. This was also clear from the interviews, as all participants found at least some risk in rejected contributions. The most common reason for a rejection was a misalignment between the feature that a company wanted – or had already developed – and the direction the maintainers wanted the project to go in. Linåker and Regnell (2020) found this risk as well. Sometimes there will be no solution for this problem. However, there are mitigation steps that can be taken. Communicating what you want to develop, explaining the reason it should be added and showing a proof of concept is most likely done with internal stakeholders already. Expanding this to maintainers and other members of the open-source community provides valuable feedback on

whether a feature will be accepted. If it is not, this feedback and subsequent discussion will steer the development into a direction that is acceptable for both parties. Further down the road, when a relationship with the maintainers has been built by multiple accepted contributions, this process will become easier as trust between the maintainers and company grows. For projects that are particularly important for the company, it can be worth it to attempt to have an employee become a maintainer, but it should be noted that this does come with responsibilities as well as benefits. Additionally, if the project is small, it may be a risk to the existence of the project if the employee leaves or their role in the company changes. It should be decided ahead of time how this will be handled. Introducing a successor to the project early can prevent this issue. The second reason contributions can be rejected, which is a novel finding, is that maintainers may not want to review it at all. They have no obligation to look into a feature. Reasons to refuse a review could be that the feature that is presented is too specific to the company, its use case not clear, the amount of code too large or the contributor unknown and uncommunicative. These issues can also be avoided by communicating early and professionally, again proving its importance.

Licenses

While license risks do exist, these risks should not be difficult to prevent. These risks are well known, licenses are an important part of the open-source software landscape. The risk that software has to be made publicly available due to the use of copyleft licenses has a high impact, but it is also prevented by avoiding the use of software that is under these types of licenses. On the other hand, the fear that contributed software can be used freely by anyone – without them having to open up their software – can be avoided by only using copyleft licenses and avoiding permissive licenses. Both can be enforced by tools that check the licenses of any open-source project used in development of new software. Augmented by the education of developers and other stakeholders on the importance of proper license compliance should mitigate the risk of using the wrong licenses.

Intellectual Property

Intellectual property protection is another high-impact area. This can be an issue for companies who generate revenue from their product portfolio. However, it can also mean a loss of competitive advantage for service based companies. Both risks occur when code was contributed or information exposed when it should not have been. The decision whether or not to contribute a feature should be considered carefully, as discussed by Linåker and Regnell (2020) and Munir et al. (2016) as well. Generally, the benefits outweigh the costs of keeping code private more often for product based companies, as they can generate revenue from it directly. It can also happen that mistakes by employees or forced contribution due to a license cause code to be contributed that should

not have been. Both of these risks are be mitigated by clear communication on contribution guidelines, education of developers and other stakeholders and internal code reviews.

It is also important that ownership of intellectual property is established in contracts with employees – and with business customers if applicable. This was a novel finding. While not many interviewees mentioned this risk, the ones who did stressed that this can be a high impact area, as it can potentially cause conflicts between the company and its employees. Especially if contributors were hired for their work on open-source projects that a company finds interesting, the boundary between contributions as employee and contributions as a private individual can be hard to define. Continuous communication between employees who contribute in their own time and the company is good practice to stay ahead of issues like this.

Brand Image Damage

A novel finding was that contributing to projects that are themselves controversial – or run by a controversial company or person – can cause damage to a brand. This may occur if the project makes questionable partnership choices, or people heavily involved become controversial. Proper vetting of projects and the people involved, before contributing and periodically while working with the project, can prevent issues in this area.

Another novel finding was that improper conduct in communities poses a risk to the brand image. Even though this is not an issue of damage to the brand in general, a bad reputation inside open-source communities can have a direct negative impact. Poor communication, demanding too much or not contributing can lead to a poor relationship with maintainers and the community. This in turn leads to less goodwill when contributing features or asking for support. Educating employees on proper conduct in open-source communities and contributing regularly is vital to build and maintain a mutually beneficial relationship.

Linåker and Regnell (2020) found that companies fear brand image damage as a result of misuse of the software the company has released as open-source. However, during this research this risk was not found. They did consider company owned projects as well in their research, while the scope of this research only concerns contributions to projects that are not owned by the company. As contributing companies are less visible than the owner of an open-source project, the risk may be lower or nonexistent.

Other Risks

In a vacuum, contributing a feature comes with additional costs. It takes time to generalise the feature, adapt it to the standards of the community and discuss it with maintainers. Then, waiting for the review and potentially redoing some work takes time as well. AlMarzouq et al. (2005), Butler et al. (2019), and Linåker and Regnell (2020) also found these risks. However, this is not a com-

plete view of the situation. While developing a single feature may incur costs, other members of the community are working on features that do not cost the company anything. This saves the company the cost of having to develop those features. Furthermore, the company does not have to pay an upfront or licensing fee for the software. The additional costs can also be lowered by developing as if any change to the code will be contributed. Although it does not reduce the time spend on discussions, it reduces the costs of generalisation. If the company can then also deploy the feature they developed with a temporary fork, it does not matter that they have to wait until their feature is integrated into the main project. For this to work, the company may need to invest in changing their development process to fit with open-source development. However, these are one-time costs. Finally, and as discussed before, keeping features private can incur much higher costs over time. Even if the company decides that a feature should not be contributed, the fact that it is developed and generalised to fit with the project as closely as possible, means that it can be kept closer to the main project and thus save maintenance costs.

Security topics yielded less risk than expected. As Linåker and Regnell (2020) found as well, there are fears that contributing code to open-source projects exposes security vulnerabilities. However, with proper internal reviews, security tools and training for employees, this risk can be reduced. Additionally, the fact that the maintainers of a project review the code and more members of the community look at it as well, means that any flaw that has been introduced can be caught more quickly. The other potential issue is exposing information or data. This risk can be reduced by the same mitigation strategies. All in all, there should not be more security risks concerned with contributing code than there are with keeping the code private.

Some novel risks around working with open-source software for business customers were found. Discussing these potential issues ahead of time with the customer will mitigate most of these risks. If the business customer wants more control over contributions, they and the company should stay in contact during development so potential contributions can be discussed on a case-by-case basis.

Finally, there is an interesting finding compared to Linåker and Regnell (2020). They found that not contributing to a project may have a negative impact on its health, as it needs contributors to survive. On the other hand, interviewees in this research also warned about the opposite: projects becoming too dependent on the contributions by the company. While this sounds like a contradiction, both can be true. A company has to be careful and find a balance between contributing to a project to keep it healthy and preventing it from becoming too dependent on the company.

Mitigation

Looking at the bigger picture, some general mitigation strategies can be employed. Communication about open-source contribution inside the company should be clear and continuous. For some projects, it may be necessary to discuss each potential contribution, while for some projects this is not necessary.

Individual developer should have a point of contact to discuss contributions. This can either be a direct superior or a contact at a separate part of the company, such as an open-source program office. Communication with the open-source projects that a company works with is important as well. As mentioned before, this can reduce multiple risks. It has the additional benefit of building relationships and building influence. Educating employees on these and other mitigation strategies is vital for these processes to work well. Creating policy is important, but without educating employees on how it works and what its benefits are, the policy will not provide benefits. Finally, noninvasive tooling to support the policies and other risk mitigation strategies noted earlier can be a very valuable resource. This is especially true for high impact risks that are easy to mitigate, such as mistakes with licenses and information leaks.

While the the review process was not mentioned by interviewees as a general risk mitigation strategy, it is an important facet of risk reduction in multiple categories. While it is common in software development processes to perform code reviews, reviews for open-source contributions add some dimensions. The review should consider if the code is up to the standards of the specific community. It should check for potential information leaks and intellectual property that should be kept private or that should be patented. Finally, the review should consider the license of the project and if contributing to that project fits with the company guidelines on licenses. If that is done properly, multiple risks can be reduced by performing reviews.

Final Considerations

The results indicate that contributing is almost always the correct choice. Only in a situation where keeping the code private has a clear monetary or competitive advantage, it may be the correct choice not to contribute. Even then, it may be the case that the costs are higher than the gains in the long run if investing in the maintenance effort is more expensive.

Additionally, companies should consider the ethics of open-source. It is allowed under some licenses to use the code in any way the user wishes, including taking it private and selling it. However, unless there is some clear benefit to keeping the code internal it is both ethical and beneficial to contribute. Withholding code that costs the company money from others in the community who may be able to benefit from it goes against the principles of open-source software. For restrictive licenses, it is clear that contributing is expected and may even be required.

The question that remains is why companies do not always contribute code even if it is more beneficial to open-source it. The most likely reason is a fear or misunderstanding of open-source software contributions. As discussed, companies do see risks around contributing. What they may not see, is that many of these risks are either low-impact, such as brand image damage, or have straightforward mitigation strategies, such as licenses and intellectual property risks. Another point of doubt may be the loss of control. When a feature is contributed and accepted it is then part of the open-source project and will

be treated as such. This means the company itself has no way to control its development direction, unless it has managed to build considerable influence in the community. However, this loss of control is a fallacy, as not contributing a feature also means a loss of control. A company that does not contribute cannot change the development direction of the project and could end up in a situation where their changes no longer work with the main project code base. This is unpredictable and could be a big issue if the features are fundamental to the work of the company. It then has to decide to scrap the features or stay on an older version of the project. This fork of the project then becomes an additional code base that the company has to maintain.

One part of risk mitigation that has not been mentioned often, but that potentially has a big impact: start small. It can be daunting to see the necessary process changes and commitments. Small, low-impact contributions to open-source projects may be a way of easing into these processes. Taking a chance on a smaller piece of software or a bug fix to become familiar with the process of contributing can help build trust in the process. Then, increasing the size and amount of contributions slowly can form some idea of the time necessary to contribute features. This process will become easier and less time-consuming when experience is gained and a relationship with the project maintainers is formed. Taking these smaller steps before committing to drastic process changes can reduce the fear of other stakeholders in the company as well.

5.2 Answering the Research Question

The research question as defined in Chapter 1.5 is: *How can companies mitigate the risks of contributing to open-source software projects they do not control?*

To answer this question, interviews were done with experts in the field. Analysis of these interviews yielded nine risk categories, each existing of multiple different risks. For each of these risks, one or more mitigation strategies has been found, as visualised in Figures 1 - 9. Risk mitigation strategies that can be applied in general were discussed as well and were visualised in Figure 10.

Finally, some risks and mitigation strategies were particularly impactful. These were discussed earlier in this chapter. The most impactful risks and mitigation strategies can form the basis of a contribution policy for companies. The less impactful risks and their mitigation strategies can be considered on a per-company basis. Some will apply and should be mitigated. For others, the low impact can be leveraged as a way to lessen the fear around open-source contributions.

As discussed in this chapter, some findings validated earlier research. The work by Linåker and Regnell (2020) informed part of the research and interview direction of this thesis. As they note in their conclusion, further generalisation and validation of their finding was needed, which this thesis partly provides. Furthermore, some novel findings were contributed to the field. These findings broaden the view of risk and risk mitigation strategies for companies that want to contribute to open-source projects. Finally, this thesis argues in stronger terms than earlier research that contributing code that has been developed is in

most cases more valuable than keeping that code private. While this is under the condition that the risk mitigation strategies presented in this thesis are employed properly, this hopefully incentivises companies to contribute more to open-source software.

6 Conclusion

6.1 Application in practice

In practice, not all risks that were discussed in Chapter 4 will be relevant for every company – most clearly the risks around business customers, as these are only relevant for consultancy firms and similar companies. Following this, not all risk mitigation strategies are necessary to implement. In fact, even for those risks that do apply, it may not be desirable to implement all mitigation strategies that could reduce that risk. Every company should consider the costs and implications of implementing each strategy. For example, requiring internal permission and review for every communication does reduce the risk of leaked information, but it is also a measure that slows down the development process and that may irritate employees.

As discussed in Chapter 5.1, there are some risks and mitigation strategies that were generally found to be high impact. For a company that wants to start contributing to open-source projects, this is a good starting point. However, it is still good practice to consider all risks and mitigation strategies that were discussed in this thesis before contributing, as some of the risks not mentioned here can still have an impact.

Licenses

A choice should be made on which licenses will be allowed. In general, it should at least be decided whether highly restrictive, restrictive, permissive licenses, or a combination of the three are allowed. This choice could be made for each project separately, or as overarching company policy. Both the advantages and disadvantages of these license categories discussed in Chapters 2.3 and 4.8 should be taken into account. Once this choice is made, it may be valuable to decide whether all or some subset of the licenses that fall under each allowed category are to be used. There may be slight differences between licenses that qualify one while disqualifying another, even within these categories. However, restricting access to more licenses does mean restricting access to more – potentially useful – projects. A balance has to be found between restricting licenses that bring risks and unintended consequences with the flexibility of choosing projects. Additionally, these choices should be considered even when only consuming open-source, as highly restrictive licenses already impact development processes if proprietary software is linked to it. Finally, it is possible to make different choices about which licenses are allowed for consumption and which licenses are allowed for contribution. If this is the case, it should be made very clear to developers who might want to contribute, as it could be confusing.

Intellectual Property

Deciding how strict intellectual property protection will be enforced should depend on the source of revenue of a company. If it is a service based company, it can be much less strict, as intellectual property is not its main source of income.

However, a product based company should enforce much stricter rules to make sure profitable intellectual property is not shared accidentally. Such a company could benefit from internal reviews of pull requests not only to find mistakes and ensure its quality, but also to decide on whether a piece of software could instead be monetised. Similarly, clear guidelines supported by tooling to sanitise code and comments that are to be contributed to remove any potential leaks of proprietary information can be valuable as well.

Development

To fully benefit from contributing to open-source projects, some changes should be made in the development process. First, it is good practice to develop any feature or bug fix as if they will be contributed to the project. This includes reviewing the code with the standards of the project in mind, but also checking for valuable intellectual property and license compliance. Not only will this save time if it is indeed decided that it will be contributed, if it is kept internal instead it will fit with the project as closely as possible. This reduces the cost of keeping a fork up to date. Working in this way also reduces the chance that unexpected costs are incurred because the code has to be rewritten to fit with the project requirements.

Second, as it is possible that features that are contributed will be added in an upcoming version release of the project and will not be integrated immediately, it is important to have a way to deploy the current version including the feature that was just contributed. This does mean maintaining an internal fork for some time, but this fork should become obsolete when a new version of the software is released, thus not incurring the continuous costs that come with maintaining a fork indefinitely. In return, any feature that is developed can be used immediately.

Finally, encouraging developers to contribute whenever possible ensures that no unnecessary internal forks or patches are created. As the path to not contributing is often easier than doing the work of contributing, while it potentially costs more to maintain it in the future, the company should try to persuade developers that contributing is the correct choice. Interviewees were very clear that it is generally the correct choice to contribute. If a decision is made that contributing is not viable, it should be a deliberate choice that is weighed against the future costs of maintaining that piece of software internally.

Communication

Communication is key in navigating open-source. Before development of a feature or bug fix is started, it should be communicated to the community and project leadership what is planned. They can then propose adjustments to the proposed plans such that it fits with the project and its development direction. During development, communicating with the maintainers and others in the community can be a valuable resource to get help or steer the development in the expected direction. Keeping these communication channels open reduces

the risk of further work down the line, or total rejection of a contribution. Especially the latter comes with the cost of maintenance in perpetuity and should be avoided.

For each project, the choice should be made to what extent the company wants to be involved. To reduce the risk that contributions are rejected and to exert influence on the direction of development, interviewees recommended to become more ingrained in the community and to have developers become maintainers in the project if possible. This is an investment of resources that should be weighed against the benefits. Increasing communication efforts by discussing features from others in the community and joining discussions on the project in general are intermediate steps that could be taken to become more active and well known in the community.

Inside the company itself, having clear communication channels to ask questions of those who decide whether contributions can be made is important as well. If these channels do not exist or are not clear, it is possible that developers will choose to forego contributing, which leads to creating forks and patches that need to be maintained, as discussed before. It can be valuable to create an Open Source Program Office to fulfil this task, among others.

Policy and Tools

The choices that were made and rules that were defined about the topics above, should be established as policy. This means that is should be clear to all stakeholders what these decisions are, and why they were taken. This policy should also be available to reference for employees who need the information.

This policy can, and in some cases should, be supported by tools. For example, a library manager that lets the user know that a library they want to use is under a license that cannot be used in their project could save time and mitigate a risk. While it may be tempting to reduce as much risk as possible by using tools, this does reduce flexibility and speed in the development process. A balance has to be found, which may require continuous monitoring and readjustment to find the best setup for the company.

Education

Properly executing these policies does require education and training initiatives. The development process of open-source projects differs from proprietary, in-house development, so introducing the required concepts to developers is necessary. The same is true for other processes that are required of developers who interact with open-source. Proper communication and expectations management, license awareness, and rules around sharing or not sharing intellectual property are most obviously areas that should be considered for this.

6.2 Limitations

There are some threats to validity in the design of this research. First, there may be sampling bias. Although sampling was fairly random – any response from Discord servers and any response from the LinkedIn search was accepted if it fit the criteria – there can be self-selection bias in the sample. In both cases, a large group was asked to join the study, but it is possible that people with a vested interest in open-source or strong opinions on the topic have a higher probability of accepting. Considering the nature of open-source software, a topic people can be very enthusiastic about, the chance that those who responded were very positive is large. To counteract the effects of this, extra care was taken during interviews and afterwards during analysis of the interviews to look out for answers that were only positive or downplayed risks to an unreasonable level. None of the interviews featured these types of responses.

Another limitation is the reliance of interview questions on earlier research. This choice reduces discoverability of risks that may not have been found. To mitigate this, interviewees were asked about risks in general first. However, this does produce the most obvious answers. Another question at the end of the interview whether there was anything else the interviewee had to add did lead to some additions.

Finally, the results as presented may skew the view of the underlying impact of the risks or mitigation strategies. As can be expected from qualitative research into risks, the most obvious risks mitigation strategies will be mentioned most often in interviews. On the other hand, risks that occur less often but have a high impact will not come up as often. In the visual representations, this nuance is lost. To counteract this, the impact of risks as noted by the interviewees is taken into account and emphasised in the text of the results and especially the discussion.

6.3 Future Work

Future work should look into quantifying the risks found in this research. One aspect that would be particularly interesting is the development of a model to calculate the future cost of maintenance required for internal forks and patches. This turned out to be a big risk factor. Estimating these costs and comparing it to the benefits of keeping it private would help in analysing the choice of whether or not contributing a piece of software is worth it.

Building influence in a community was mentioned multiple times as a way to reduce the risk of features getting rejected and the project moving in a direction that is not in the best interest of the company. While steps that can be taken are known and have been discussed, it would be interesting to research how much of an investment it is to build this influence. Additionally, finding to what extent this influence actually reduces the mentioned risks would be interesting to know as well.

Finally, while each risk category deserves more individual attention, two categories emerged that are particularly interesting for future research. The

sustainability of projects, especially smaller projects, was found to be a risk factor that has not been discussed in detail in earlier research. Included in this topic is the question of why companies apparently do not sponsor projects that are important to their work, even if the maintainers of the project work with constrained resources. Additionally, it would be interesting to find what can be done to make projects with a small team less prone to failure if developers leave.

Total rejection of contributions was seen as a risk by all interviewees. While ways to reduce the risk of rejection were discussed in Chapter 4.5, this is a complex topic that could be expanded upon. It would be an interesting research avenue to discover exactly what reasons for rejection are, besides those mentioned in this research, and how developers can navigate these issues.

A Interview Questions

Here, the list of interview questions used in the semi-structured interviews is presented. The purpose of each question and potential follow-up questions is explained.

What is your current job position?

Determine source of expertise of the interviewee and context for their answers.

How many people does your company employ in software development (or similar)?

Determine context for the answers of the interviewee.

Could you describe the type of software your company generally develops?

Determine context for the answers of the interviewee.

Does your company make use of Open Source Software? For what tasks?

Determine context for the answers of the interviewee.

Does your company contribute to these OSS projects?

Establish whether the interviewee's company contributes to open-source software projects and moving towards the topic of the interview.

Does your company also have its own OSS projects? (If so, please leave those out of consideration for the next questions)

Determine context for the answers of the interviewee. Make sure the interviewee understands the direction of the research.

Does your company contribute bug fixes to OSS projects?

Does your company contribute features to OSS projects?

Does your company contribute to OSS projects in other ways? (documentation, monetary, etc)

Different ways of contributing may carry different risks.

Do you think your company faces risks when contributing to OSS? What risks does your company face when contributing to OSS? How does your company mitigate these risks?

Determine what the interviewee thinks of as risks when contributing to open-source software. This open question is asked first as to not lead the interviewee into a certain direction with questions related to earlier research.

Are these risks different for contributing bug fixing compared to contributing features? And other types of contributions? (if applicable)

Determine whether there is a difference in risk between the different ways of contributing.

More specifically, does your company encounter...? And how does it handle...?

- Risks of intellectual property loss?

- Risk of misaligned priorities between company and project?

- Risk of brand image damage?

- Risk of exposing security vulnerabilities?

- Risk due to licenses?

- Risk of extra costs due to increased development time?

- Risks that come with not contributing?

Direct questions into risks that were found from earlier research (Chapter 2). For each of these questions, the main goal is finding the ways companies use to mitigate these risks.

How does your company mitigate these risks?

A more general question into risk mitigation. While specific risk mitigation strategies should have emerged in the questions before this one, anything that has been left out due to the specific nature of those questions should emerge here.

Is there a difference between company-owned OSS projects vs non-company-owned OSS projects regarding risk or how risk is reduced? (If applicable)

Find out if there is indeed a difference between contributing to projects that the company does not control or own, and contributing to projects that the company does control or own.

Is there anything I missed?

If there anything that the interviewee has thought of during the interview that has not come up through a question, they can mention it here.

B Codebook

Category\Subcategory	Code	Frequency	Nr. of interviews
brand_image\brand_image_benefits	contributing_seen_as_positive	6	6
brand_image\brand_image_benefits	employee_benefits	6	6
brand_image\brand_image_risk	brand_image_damage	11	9
brand_image\brand_image_risk	connection_to_projects	3	3
brand_image\brand_image_risk	incorrect_support_answers	1	1
brand_image\brand_image_risk	introduce_security_vulnerability	2	2
brand_image\brand_image_risk	low_quality_contributions	6	4
brand_image\brand_image_risk	unprofessional_conduct	6	5
brand_image\brand_image_risk_mitigation	employee_consequences	2	2
brand_image\brand_image_risk_mitigation	internal_code_review	3	2
brand_image\brand_image_risk_mitigation	professional_communication	4	4
brand_image\brand_image_risk_mitigation	vet_projects	2	2
brand_image	low_quality_contribution_no_risk	5	4
business_customers\business_customer_benefits	b2b_marketing	11	7
business_customers\business_customer_benefits	customer_wants_support	3	3
business_customers\business_customer_risk	b2b_contract_risk	2	2
business_customers\business_customer_risk	contribution_hours_unpaid	2	1
business_customers\business_customer_risk	customer_wants_proprietary	4	3
business_customers\business_customer_risk_mitigation	choose_customers_who_want_contribution	2	1
business_customers\business_customer_risk_mitigation	discuss_with_customer	7	4
business_customers\business_customer_risk_mitigation	dont_fix_the_issues	1	1
business_customers\business_customer_risk_mitigation	give_customer_discount	3	2
business_customers\business_customer_risk_mitigation	invoice_hours	1	1
business_customers	low_customer_involvement	1	1
contribution_strategies	bug_discovery	3	3

contribution_strategies	client_takes_contribution_credit	1	1
contribution_strategies	code_review	2	2
contribution_strategies	community_engagement	4	3
contribution_strategies	documentation_contribution	8	7
contribution_strategies	event_contribution	5	3
contribution_strategies	give_devs_OS_time	4	4
contribution_strategies	give_project_to_foundation	1	1
contribution_strategies	maintain_projects	1	1
contribution_strategies	make_software_compatible	1	1
contribution_strategies	monetary_contribution	6	6
contribution_strategies	support_contribution	3	3
financial\financial_benefits	community_maintains_contributions	4	4
financial\financial_risk	contribution_integration_takes_time	6	4
financial\financial_risk	extra_development_time	8	6
financial\financial_risk	infrastructure_cost	1	1
financial\financial_risk	opportunity_cost	2	2
financial\financial_risk_mitigation	fix_internal_process	1	1
financial\financial_risk_mitigation	increase_awareness	1	1
financial\financial_risk_mitigation	use_linter	1	1
financial	gain_higher_than_cost	6	5
general_mitigation	account_management	3	2
general_mitigation	bug_feature_risk_difference	5	5
general_mitigation	clear_guidelines	21	9
general_mitigation	communication	9	7
general_mitigation	culture	4	3
general_mitigation	developer_education	4	4
general_mitigation	full_time_project_members	1	1
general_mitigation	gather_information_on_project	2	2

general_mitigation	open_source_department	2	1
general_mitigation	outside_help	9	5
general_mitigation	specialize	2	1
general_mitigation	tooling	14	6
general_mitigation	trust_employees	2	2
intellectual_property\ip_risk	client_ip	1	1
intellectual_property\ip_risk	competitive_advantage_loss	5	5
intellectual_property\ip_risk	employee_ip	6	3
intellectual_property\ip_risk	lose_potentail_revenue	3	3
intellectual_property\ip_risk	mention_upcoming_projects	4	3
intellectual_property\ip_risk	patent_trolls	1	1
intellectual_property\ip_risk	proprietary_code	10	9
intellectual_property\ip_risk_mitigation	defensive_patents	4	2
intellectual_property\ip_risk_mitigation	delay_contribution	1	1
intellectual_property\ip_risk_mitigation	discuss_before_contribution	6	5
intellectual_property\ip_risk_mitigation	include_ip_in_contract	3	2
intellectual_property	no_ip_loss_risk	4	4
intellectual_property	no_proprietary_code	4	2
license	contributor_licence_agreement	3	3
license\license_benefits	contributions_stay_open_source	1	1
license\license_risk	bigger_company_bigger_target	1	1
license\license_risk	licence_risks	6	5
license\license_risk	lose_financial_opportunities	2	1
license\license_risk	no_license	1	1
license\license_risk	non_commercial_only_license	1	1
license\license_risk	required_contribution	9	4
license\license_risk_mitigation	choose_specific_licenses	6	6
license\license_risk_mitigation	legal_experts	6	4

license\license_risk_mitigation	license_awareness	9	7
license\license_risk_mitigation	package_manager	1	1
license\license_risk_mitigation	properly_attribute_code	1	1
license\license_risk_mitigation	review_contributions	1	1
license\license_risk_mitigation	seperate_code_bases	3	1
license	no_license_risk	3	2
not_contributing\not_contributing_benefit	cheaper	1	1
not_contributing\not_contributing_risk	brand_damage	2	2
not_contributing\not_contributing_risk	contributing_is_expected	7	5
not_contributing\not_contributing_risk	employees_like_contributing	1	1
not_contributing\not_contributing_risk	incompatability_risk	1	1
not_contributing\not_contributing_risk	less_project_understanding	1	1
not_contributing\not_contributing_risk	maintanance_cost	16	9
not_contributing\not_contributing_risk	no_community_review	2	2
not_contributing\not_contributing_risk	no_influence	3	3
not_contributing\not_contributing_risk	no_updates_for_fork	5	3
not_contributing\not_contributing_risk	slow_community_support	3	2
not_contributing\not_contributing_risk	tech_debt	8	7
not_contributing\not_contributing_risk_mitigation	build_feature_in_own_app	1	1
not_contributing\not_contributing_risk_mitigation	plugin_architecture	3	2
not_contributing\not_contributing_risk_mitigation	reconsider_contribution	5	5
not_contributing\not_contributing_risk_mitigation	reduce_tech_debt	8	5
owned_versus_external_project\benefits_external	help_quickly_accepted	1	1
owned_versus_external_project\benefits_owned	owned_company_specific	1	1
owned_versus_external_project\benefits_owned	owned_control_tooling	2	1
owned_versus_external_project\benefits_owned	owned_determine_direction	6	5
owned_versus_external_project\benefits_owned	owned_reject_contributions	1	1
owned_versus_external_project	risk_difference	4	4

owned_versus_external_project\risks_external	external_maintainer_hestitance	1	1
owned_versus_external_project\risks_external	maintainers_quitting	3	2
owned_versus_external_project\risks_external	more_conflict	1	1
owned_versus_external_project\risks_owned	community_building	6	4
owned_versus_external_project\risks_owned	responsible_party	1	1
owned_versus_external_project\risks_owned	stay_ahead_of_competitiors	1	1
owned_versus_external_project\risks_owned	urgent_issues	1	1
rejected_contributions\rejection_risk	bad_communication	4	4
rejected_contributions\rejection_risk	contribution_too_large	3	3
rejected_contributions\rejection_risk	misaligned_direction	6	4
rejected_contributions\rejection_risk	rejected_contribution	15	14
rejected_contributions\rejection_risk	unkown_developers	1	1
rejected_contributions\rejection_risk_mitigation	become_maintainer	6	6
rejected_contributions\rejection_risk_mitigation	communicate_with_community	17	10
rejected_contributions\rejection_risk_mitigation	create_fork	12	8
rejected_contributions\rejection_risk_mitigation	show_your_expertise	4	3
rejected_contributions\rejection_risk_mitigation	understand_project_needs	3	3
security	no_difference_with_proprietary	7	5
security\security_benefits	oss_better_security	6	4
security\security_risk	company_information_leak	6	5
security\security_risk	expose_customer_information	1	1
security\security_risk_mitigation	customer_info_protection	2	2
security\security_risk_mitigation	developer_attitude	2	2
security\security_risk_mitigation	healthy_projects	1	1
security\security_risk_mitigation	maintainers_review	6	6
security\security_risk_mitigation	review_before_contributing	5	4
security\security_risk_mitigation	security_tools	9	7
security\security_risk_mitigation	security_training	2	2

sustainability\sustainability_risk	projects_discontinuing	6	4
sustainability\sustainability_risk	role_change_employee	2	2
sustainability\sustainability_risk	single_maintainer_risk	3	3
sustainability\sustainability_risk_mitigation	fork_and_maintain	1	1
sustainability\sustainability_risk_mitigation	maintain_on_personal_account	1	1
sustainability\sustainability_risk_mitigation	multiple_maintainers	2	2
sustainability\sustainability_risk_mitigation	pay_maintainers	2	2

References

AlMarzouq, M., Zheng, L., Rong, G., & Grover, V. (2005). Open source: Concepts, benefits, and challenges. *Communications of the Association for Information Systems*, *16*(1), 37.

Andersen-Gott, M., Ghinea, G., & Bygstad, B. (2012). Why do commercial companies contribute to open source software? *International journal of information management*, *32*(2), 106–117.

Butler, S., Gamalielsson, J., Lundell, B., Brax, C., Sjöberg, J., Mattsson, A., Gustavsson, T., Feist, J., & Lönroth, E. (2019). On company contributions to community open source software projects. *IEEE Transactions on Software Engineering*, *47*(7), 1381–1401.

De Laat, P. B. (2005). Copyright or copyleft?: An analysis of property regimes for software development. *Research Policy*, *34*(10), 1511–1532.

Fitzgerald, B. (2006). The transformation of open source software. *MIS quarterly*, 587–598.

Henkel, J. (2006). Selective revealing in open innovation processes: The case of embedded linux. *Research policy*, *35*(7), 953–969.

Kallio, H., Pietilä, A.-M., Johnson, M., & Kangasniemi, M. (2016). Systematic methodological review: Developing a framework for a qualitative semi-structured interview guide. *Journal of Advanced Nursing*, *72*(12), 2954–2965. https://doi.org/https://doi.org/10.1111/jan.13031

Kendall, J. E., Kendall, K. E., & Germonprez, M. (2016). Game theory and open source contribution: Rationale behind corporate participation in open source software development. *Journal of Organizational Computing and Electronic Commerce*, *26*(4), 323–343.

Lerner, J., Pathak, P. A., & Tirole, J. (2006). The dynamics of open-source contributors. *American Economic Review*, *96*(2), 114–118.

Lerner, J., & Tirole, J. (2005). The scope of open source licensing. *Journal of Law, Economics, and Organization*, *21*(1), 20–56.

Linåker, J., & Regnell, B. (2020). What to share, when, and where: Balancing the objectives and complexities of open source software contributions. *Empirical Software Engineering*, *25*(5), 3799–3840.

Lundell, B., Butler, S., Fischer, T., Gamalielsson, J., Brax, C., Feist, J., Gustavsson, T., Katz, A., Kvarnström, B., Lönroth, E., et al. (2021). Effective strategies for using open source software and open standards in organizational contexts: Experiences from the primary and secondary software sectors. *IEEE Software*, *39*(1), 84–92.

Merriam, S. B., & Tisdell, E. J. (2015). *Qualitative research: A guide to design and implementation*. John Wiley & Sons.

Morgan, L., & Finnegan, P. (2007). Benefits and drawbacks of open source software: An exploratory study of secondary software firms. *Open Source Development, Adoption and Innovation: IFIP Working Group 2.13 on Open Source Software, June 11–14, 2007, Limerick, Ireland 3*, 307–312.

Munir, H., Wnuk, K., & Runeson, P. (2016). Open innovation in software engineering: A systematic mapping study. *Empirical Software Engineering*, *21*(2), 684–723.

Nagle, F. (2019). Open source software and firm productivity. *Management Science*, *65*(3), 1191–1215.

Open Source Initiative. (2007). *Open Source Initiative: the open source definition*. Retrieved August 15, 2023, from https://opensource.org/osd/

Oreg, S., & Nov, O. (2008). Exploring motivations for contributing to open source initiatives: The roles of contribution context and personal values. *Computers in human behavior*, *24*(5), 2055–2073.

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