Universiteit Leiden
ICT in Business and the Public Sector

Assessing Risks of Open Source Components in Software Due Diligence

Name: Age Kruijssen
Student-no: s2281457

Date: July 10, 2020

1st supervisor: Joost Visser
2nd supervisor: Xishu Li

MASTER'S THESIS

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

Abstract

Background: The majority of commercial software uses Open Source Software (OSS) components. In acquisitions of software-intensive companies there is growing need to understand and assess OSS component risks.

Objective: To assess risks of OSS components we built a model and supporting tool. This tool helps in bridging the gap between qualitative and quantitative risk assessments by using data from different sources to provide recommendations on risk mitigation actions.

Method: Our research consists of an exploratory and a confirmatory part. To create an understanding of value adjustments for OSS component risks, exploratory research is done through literature research, followed by the development of a model through literature findings and interviews, and concluded by case studies. We then confirm the validity of our model by creating a model implementation which is applied in case studies.

By studying risks of OSS component usage described in literature, we found that the risks can be split into two main categories: security and copyright compliance risks. For each category, one qualitative model was developed which shows how identifiable situations are related to risk occurrences, and how these situations can be measured. We validated and completed these models on the basis of expert input. Five experts in software risk assessment and valuation were identified whom were split into two groups. One group was used to validate and add to the literature findings. These findings improved the qualitative models after which the improved models were validated with the second group.

The qualitative models were implemented in a tool to assess OSS component risks. Three Software Due Diligence (SDD) cases were identified for which OSS component risk data is available. The tool was applied and improved through these case studies to validate and improve the understanding of how risk assessment can be used in the value determination of software.

Results: We found seven measurable indicators, and seven situations, to identify security risks. For copyright compliance we found three measurable indicators and nine situations. Our tool was able to identify whether a component was outdated, had vulnerabilities, contained a copyleft license or no license at all and was able to identify whether a component was actively supported or not.

Conclusion: Our models and supporting tools can be used by SDD advisors to support their clients in quantifying and mitigating risks in software systems deriving from OSS component usage.

Acknowledgements

Due to the outbreak of COVID-19 my time writing this thesis took an unexpected turn; from working in an office to working from home. Nevertheless, many remarkable people have given me much (remote) support during my journey writing this paper.

First of all, I want to thank my supervisor, Joost, for his guidance, enthusiasm about the topic, and his many good points of feedback. His interest in my work inspired me to keep expanding my horizon surrounding the topics I was researching and became interested in.

Secondly, I want to thank my work supervisor, Erik, for giving me the opportunity to work in a close team within EY. Even in times of Corona, he made me feel part of the team at all times. Furthermore, I got trusted to create an extension to an existing business process and apply my acquired knowledge in a lasting product for the company.

I want to thank my colleagues at EY for always being there to brainstorm over my ideas. This helped me tremendously in finding new ideas and approaches.

I want to thank all of my family for always being there, and always support me no matter what.

Finally, I want to thank my girlfriend, Katerina, for being my guiding light, keeping me on track like no other.

Thank you!
Age

Contents

List of Figures

List of Tables

List of Abbreviations

TDD Technical Due Diligence . . . 1

SDD Software Due Diligence . . . 1

OSS Open Source Software . . . 1

EY Ernst & Young . . . 1

M&A Mergers & Acquisitions . . . 58

LGPL GNU Lesser General Public License . . . 15

GPL GNU General Public License . . . 15

AGPL Affero General Public License . . . 15

LoC Lines of Code . . . 19

KLoC 1000 Lines of Code . . . 19

BPMN Business Process Model and Notation . . . 32

CVE Common Vulnerabilities and Exposures . . . 51

NACE Statistical Classification of Economic Activities in the European Community (Nomenclature statistique des activités économiques dans la Communauté européenne) . . . 38

TF Technology Factor . . . 19

SS System Size . . . 19

RF Rebuild Factor . . . 19

PERT Program Evaluation and Review Technique . . . 40

RV Rebuild Value . . . 19

TI Technical Interest . . . 20

1

Introduction

According to the IT industry outlook 2020 (Computing Technology Industry Association, 2019, p.3,16,19) report, the information technology industry is projected to grow with 3.7% to a worth of $5.2 trillion in 2020. 12% of this worth accounts to software, meaning software is expected to be a $624 billion market in 2020.

Due to the growing size of the market there is an increased need for understanding software value from a business perspective. Acquisitions, mergers and other investments in companies that rely heavily on software, need to be understood in order to estimate the risks and returns that come from these investments. Technical Due Diligence (TDD) exists to look at risks on a technical level. But TDD only looks at software from a high-level perspective. This means it focuses at general software aspects such as product development, service delivery and security. Software Due Diligence (SDD) fills the low-level technical gap by evaluating software quality and the related financial implications. This includes risks from Open Source Software (OSS) components, rebuild costs and technical debt.

As part of their TDD service offerings, Ernst & Young (EY) offers SDD services. In SDD services there is a need for an extension of existing techniques to assess risks in software systems deriving from OSS component usage.

1.1 | Problem Statement

OSS components are used in 95% of commercial software (Franch et al., 2013, p.250). These components are used to add functionality to software while being maintained by an 'open source community', a community of developers that work on, among other things, open source projects. By being used in such a widespread way, open source software is an important aspect in software valuation. As the value of software is not only driven from its functionality but also from its future proceeds and costs. Costs and proceeds are related to the risks and opportunities of software, of which OSS is an important aspect.

Using OSS components in software systems comes with potential future costs through risks such as: risk of infringement of licenses and/or intellectual property, risk of software deprecation and security risks (Ruffin & Ebert, 2004; Franch et al., 2013; Hauge, Cruzes, Conradi, Velle, & Skarpenes, 2010). Risks can reduce value by the caused uncertainty in future cash flows (McKinsey, Copeland, Koller, & Murrin, 2000). However, according to García García and Alonso Magdaleno (2013) current accounting frameworks are not able to estimate the costs of OSS in these valuations. Furthermore, earlier research fails to create a holistic understanding of risks in OSS component usage and how these risks relate to value reductions in software value assessment. This understanding is essential for estimating the potential value reductions by using OSS components in value based software engineering. With a growing focus on value delivery and value estimations (Dingsøyr & Lassenius, 2016), researching the risks of OSS components and how these can be identified will add further knowledge to these areas. As services such as SDD and methods such as agile software development, continuous integration and continuous delivery are growing, the need for an understanding of OSS component risks in value determination becomes essential.

1.2 | Aims and Objectives

This research aims to fill the gap between OSS component risks and how this impacts the value determination of software. The objective is to provide tool

assistance that allows professionals engaged in value determination of software to combine various data sources to identify risks and translate those risks to value adjustments in a consistent and efficient manner.

We will research OSS component usage risks in literature and make these identifiable through the creation of risk models. These models will be implemented in a tool which will be able to analyze OSS components by leveraging data from different sources and so being able to identify the risk properties set forth in the risk models. This tool will be created as an extension to an existing business process of EY to allow professionals to engage in the risk assessment of OSS component risks in a consistent and efficient manner. By collecting data from existing SDD cases where OSS component risk data is available we will extend our risk assessment approach to include value adjustments based on OSS component risks. We will create the decision logic to recommend mitigation strategies based on the risks found in the risk assessment.

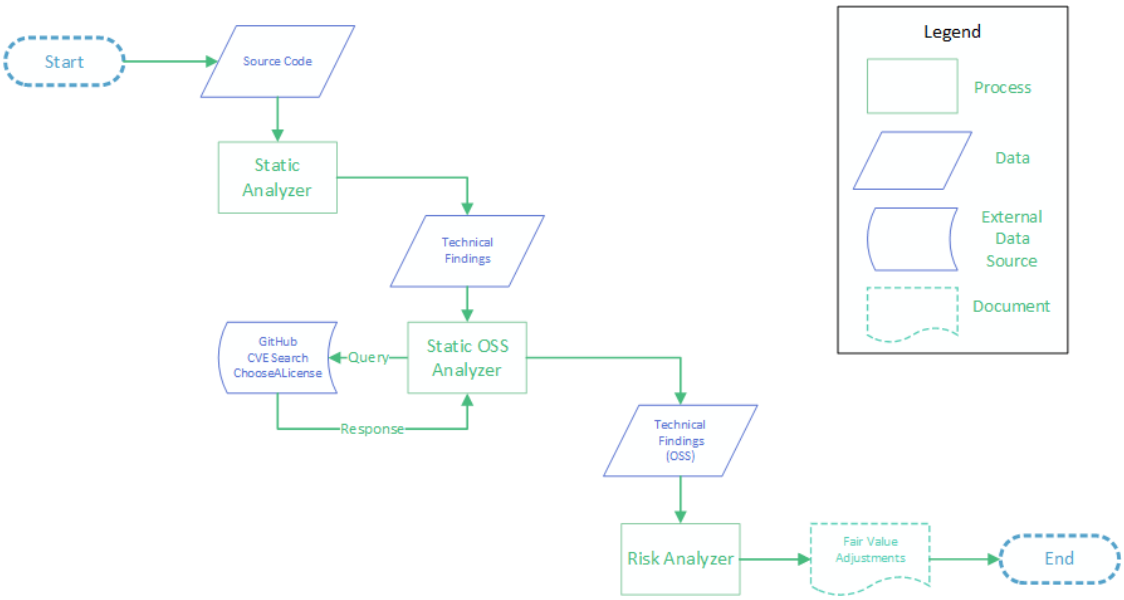


Figure 1.1: Data and process flow of the risk assessment for OSS component risks.

The OSS component risk assessment approach is shown in Figure 1.1. The *Static Analyzer* which is used to analyze the source code is already in place in the business process of EY. An extension to the business process will be made by creating an automated and consistent approach to the *Static OSS Analyzer* and

Risk Analyzer processes. The supporting tool will be able to process the output of the *Static Analyzer*. This will analyze all components that are used in the applications of a client. As part of the analyses, external sources will be be queried to create an understanding of risk in the components. The information that is gathered will be analyzed based on different risk identifiers that are identified in the risk models. These findings will be further analyzed in the *Risk Analyzer*. The analyzed risks will result in a recommendation on how to mitigate the found risks for every component. Furthermore, the risk analyzer will translate the risk assessments of the used OSS components into *Fair Value adjustments*. These adjustments can then be used in the reporting of EY towards the client.

1.3 | Scope

We focus on downward value adjustments based on risks and associated mitigation costs, while the determination of a base value based on future benefits of OSS components is kept out of scope. Determining a base value would rely on an analyzes of functional considerations and not on which OSS components are used. Furthermore, since the majority of software uses OSS components it is assumed that using OSS components is beneficial for a software company. The scope of this research is not to further identify these benefits, but rather it is to understand the other side of using OSS components: the risks.

This research will qualify the relationship between risks and goals in the usage of OSS components. A preliminary analysis will be done on the quantified assessments of risks in relation to value determination of software. As the availability on quantifiable data on this subject is limited, a statistical validation will not be possible. Therefore, this study will put forth an initial approach on how OSS component risk assessment can be used in value determination of software. This study will enable future work to improve on these assessments by using more extensive data sets.

1.4 | Research Questions

To improve the consistency of adjusting financial valuation based on the usage of OSS components the following research question has been formulated:

How can Open Source Software (OSS) Component Risk Assessment be used for the Value Determination of Software?

This question has been divided in different sub-questions. These sub-questions will help to understand the important aspects of the research question, as well as help to place the question into a larger context. To create a better understanding within the context of this study, the following sub-questions have been formulated:

1. What risks are involved in using OSS components?
2. How can OSS component risks impact fair value?
3. How can OSS component risks be identified?
4. How can the costs of mitigating OSS component risks be quantified?

1.5 | Structure

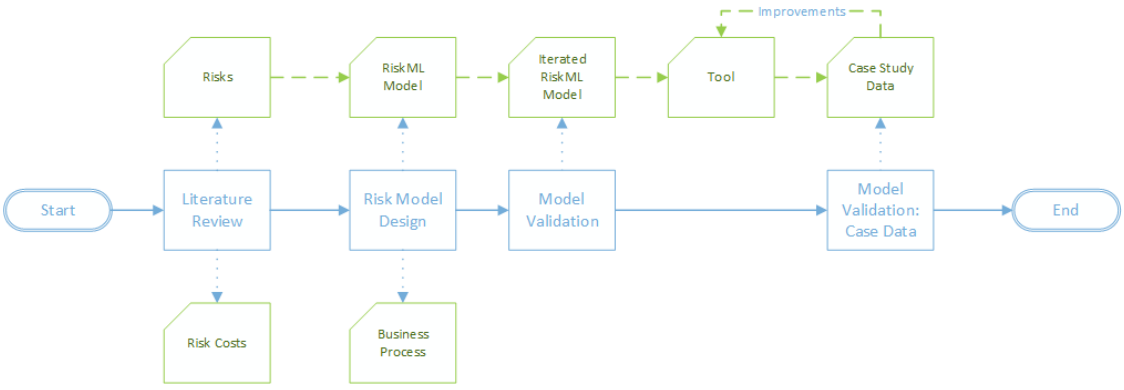


Figure 1.2: Flow of Research.

Table 1.1: Thesis Outline.

Chapter	Outline
1.	Presents an **overview** of the study: the addressed problem, its objectives, the research questions, and the scope.
2.	Provides a **literature review** of existing work. In this chapter OSS component risks, software valuation and risk costs are described based on existing literature.
3.	Outlines the methodology for the **risk model design**. This will discuss the model design based on the literature, the implementation in EY's business process, the development of the risk assessment tool and the validation of the tool through interviews and case studies.
4.	Displays the results of the **model validation** by presenting the final model based on the results of the interviews. This chapter will also present the results of the case studies.
5.	**Discusses** the threats to validity and the limitations of our research and developed tool.
6.	States the **conclusions** based on the earlier presented results and discussion. These conclusions are based on the posed research questions. Offers approaches for future works based on the limitations of this study.

This research is divided into six chapters. The flow of this thesis is shown in Figure 1.2. The content of each chapter will be described here and is outlined in Table 1.1.

The first chapter, the current chapter, serves as an introduction. Here the goal and purpose of this research are described. Furthermore, the aim, objectives and the need for this research are explained. The second chapter will present the background of the topic based on existing literature. Different concepts that support the research in this thesis will be set forth. The third chapter will introduce the process of designing the risk models: how the models are designed, validated and implemented in the existing business process of EY. Furthermore, this chapter will outline the case studies which are used to create an understanding of how risk assessment can be used in the value determination of software. The results will be outlined in chapter four which are used to validate the models. In this chapter the final models are presented based on the presented results. The literature review

and the risk model design are all part of the exploratory research. The case studies combine exploratory and confirmatory research by creating an understanding of value and by applying the models on case study data. Finally, the model validation is confirmatory research. Chapter five will discuss the threats to validity and the limitations of our research. In this chapter we will also discuss the limitations of the developed tool. In the final chapter, chapter six, we will conclude our research. Here we will describe the achieved aims and objectives, which contributions this research provides to literature as well as to EY, and how future works can further improve the done research.

Figure 1.2 provides an overview of the relationship between the different models created in this research. The first created models, which will be discussed in Section 3.2, are modelled using the RiskML modelling language. These models are based on the risks identified in the literature as discussed in Section 2.1. Improvements and validation of the RiskML models are done using unstructured interviews, of which the process is described in Section 3.4. The results of these interviews, and how this improves the models, is discussed in Section 4.2.

2

Background & Literature Review

2.1 | OSS Risks

OSS components are, just like any other software component, associated with a set of risks. Risk assessment became a topic for study around 40 years ago (Aven, 2016). Because of this the definition of risk will be discussed first. After this the different types of relevant risks will be outlined. Finally, a brief section will talk about the benefits of using OSS components in commercial software.

This section will aim to answer the following sub-question: *What risks are involved in using OSS components?* By looking at past literature on the subject a theoretical framework on the subject of risks in OSS components can be formed.

2.1.1 | Conceptualization of Risk

Kaplan and Garrick (1981, p.13) state that "risk is probability and consequence", risk is a combination of uncertainty and damage. There must be a chance of something happening and there must be a consequence to it happening. ISO (2018) defines this as a combination of consequences and the likelihood of the event happening, the "effect of uncertainty on objectives". Aven (2016) shows an overview of different conceptualizations of qualitative risks. The drawn conclusion of this research is that risk is a combination of uncertainty in relation to events and consequences. Sales et al. (2018) analyze risk in relation to value, the qualitative value that someone attributes to an object or event ('use value'). They state that value

and risk are not independent of each other: there is no risk without value attribution. If the object or event is quantifiable the relation between risk and use value also applies (Sales et al., 2018). Furthermore Sales et al. (2018) defines speculative risk and pure risk. Speculative risks have a possibility of a positive and negative outcome, while pure risks only have negative outcomes. The focus of our research is on pure risks, as using OSS components that are 'at risk' lead to a negative outcome for the user (risk damages or mitigation costs). The benefits of using OSS components are dependent on what functionality these components offer, whether other components offer similar functionality, and how the functionality may be exploited to obtain business benefits. These questions are outside of the scope of our research. Our work focuses on optimizing the understandings of the downside, assuming the upside is given.

Based on literature and the scope of this research the following conceptualization will be used:

Risk Uncertainty with the potential to have a negative effect on value.

2.1.2 | Qualitative Risks in OSS

Research by Ruffin and Ebert (2004); Franch et al. (2013); Hauge et al. (2010) provide potential risks in the usage of OSS components. The main risks relate to: risk of intellectual property right infringement, software deprecation and security risks.

Later research by Silic et al. (2015) provide a taxonomy based on past literature around the subject of OSS. Past research is combined with expert interviews to provide a complete overview. This research was done without narrowing down to OSS *components*. This research identifies the following areas of risk: general IT security risks, lack of support or ownership; confidentiality, loss of data because of malicious intent; integrity, data that becomes unreliable or inaccurate because of OSS functionality; availability, how often the system functions; performance; reliability according to specifications; maintainability, compatibility and interoperability; regulatory, license and law conformance. While Silic et al. (2015) provides

Table 2.1: Mentioned OSS risks in literature. ✓: found as an OSS *component* risk. +: found as an OSS risk.

Risk	**Ruffin and Ebert (2004)**	**Li et al. (2008)**	**Hauge et al. (2010)**	**Silic et al. (2015)**	**Sherlock et al. (2018)**	**Linh et al. (2019)**
Security vulnerabilities	✓	✓		+		✓
Code integrity				+	✓	
Reliability/Quality	✓	✓	+	+		
Forking				+	✓	✓
Changes in framework		✓				✓
Lack of Support		✓	✓	+	✓	✓
Project discontinuation			✓	+	✓	
License/copyright compliance	✓		+	+	✓	✓
License conflict with sub-components	✓			+	✓	

an extensive overview of risks in OSS, it talks specific about OSS as a whole and not about components. This makes the research less relevant for us.

As there is a lack of a complete risk overview for risks in OSS components one will be created using the literature of Ruffin and Ebert (2004); Hauge et al. (2010); Franch et al. (2013); Silic et al. (2015); Sherlock et al. (2018); Linh et al. (2019). Table 2.1 shows an overview of the findings. The risks looked at are risks for already adopted OSS components. This means adoption risks and risks not found to be related to OSS components will not be mentioned. These risks are outside the scope of this research. Risks shown in Table 2.1 have the potential to do future harm to a product or company.

2.1.2.1 | Security vulnerabilities

Vulnerabilities in components are an issue of security. By having vulnerabilities external parties can potentially harm a company through the vulnerable software. This can relate to harm such as stolen data or hostile takeover of a system. As can be seen in Table 2.1 multiple articles mention this as a risk for OSS components.

2.1.2.2 | Code Integrity

The risk of code integrity has to do with potential harm due to unwanted code execution. Since everyone is free to contribute to OSS, everyone has the ability to add unsafe code or code of low quality. Code which lowers the level of code integrity can be added with malicious intend or can be added by accident (Silic et al., 2015).

Software which contains malicious code can act like malware; potentially taking over a system, invalidate company data or giving third party access to sensitive data.

2.1.2.3 | Reliability/Quality

Reliability or quality as a risk means whether the functionality and results of an OSS component can be trusted. The risk here is the uncertainty of the changes in functionality with updates and changes to the component. According to Ruffin and Ebert (2004) OSS components should be regarded as ones own code. This means there should be no assumptions about the quality of the component, and no assumptions about a stable quality after updates to the component.

As shown in Table 2.1 this is a common risk associated with both general OSS applications and OSS components. When a component is not reliable the internal working of an application can stop functioning properly. This can cause issues such as invalid outputs or invalidation of data.

2.1.2.4 | Forking

Forking is the act of duplicating a codebase. This means that at the moment of the fork, two identical codebases exists. For example: codebase-A and codebase-B. These codebases do not share future development efforts, meaning that additions to codebase-A are not included in codebase-B, and vice versa. This can mean that one codebase is actively supported, while the other is not.

Linh et al. (2019) mention the risk of conflicting code caused by multiple people extending a project. Using multiple components forked from the same codebase could provide conflict in their functionality. Another risk is that the community that contributed to the original project is now split over multiple projects. This can decrease support causing an increase in issues such as vulnerabilities, integrity and reliability.

2.1.2.5 | Changes in Framework

Linh et al. (2019) mention the risk of changes in the core functionality of an OSS component. This means that functionality which a product can rely on might be deprecated, meaning it is no longer supported. If the relied on functionality is completely removed or altered, applications might not run properly with new versions of the OSS component.

If applications stop working in the intended way because of changes in the core functionality of OSS components, old functionality has to be maintained by the implementer of the OSS component. Alternatively components have to be replaced with a new component that provides the required functionality. In some cases rewriting certain functionality to match the updated functionality in the OSS component is enough to mitigate the risk occurrence.

2.1.2.6 | Lack of Support

Lack of support is related to risks such as reliability and security. Lack of support can be described as the inability of the provider to resolve problems (Linh et al., 2019). A component that has issues with support can increase the occurrence chance of security and reliability risks. These risks might be resolved too late, or

might not be detected at all. This puts the system at a higher risk for damages and is classified by Linh et al. (2019) as a risk with a high cost impact.

2.1.2.7 | Project Discontinuation

An important risk in OSS is that the software can be discontinued. When this happens there is no more support for the software meaning that the full maintenance responsibility comes to the implementer. As OSS projects can lack a traditional support contract as found in commercial software, there is no guarantee for the longevity of the provided support (Sherlock et al., 2018).

2.1.2.8 | License/Copyright Compliance

A license provides a contract between the licensor, the OSS owner, and the licensee, the software user. This contract gives an overview of what can, and what cannot, be done with the licensed software. The licensee is accountable for the terms described in the license of such component. With over 50 different licenses available, to fully comply with licenses it is advised to have legal council available (Sherlock et al., 2018). If no license is available the software is still protected under copyright laws (Choosealicense.com, 2019b).

The biggest issue with licenses is an issue during OSS adoption. During adoption an OSS component should be found which matches the requirements of the licensee. There are still risks of licenses in a running project. Firstly, OSS updates or forks can change the license of a project causing the licensee to no longer be compliant after this update or change. Failing to update would result in the same risks as if a project would be discontinued all together (see Section 2.1.2.7). Secondly, the requirements of the licensee change. For example, a non-distributed piece of software can become distributed. In the case of some licenses this might require open-sourcing the source-code of the software to be compliant with the OSS license, which in case of propriety software might not be what a company wants. Finally, due to the complexity of licenses a license might be wrongly interpreted. Almeida, Murphy, Wilson, and Hoye (2017) showed that most software develop-

ers have a hard time understanding licenses. This makes it questionable whether existing projects fully comply with OSS licenses.

An overview of most OSS licenses is available in Appendix A showcasing the requirements and limitation of each license. In cases of distribution the source code might have to be distributed. In some licenses this is not the case when there is an understanding of network distribution. This means for example a server running an OSS component with a license which does not define network usage as distribution, such as the GNU General Public License (GPL). When this server is called by a web-application the software is not identified as being distributed, meaning the server code can comply with the license's requirements without being open-sourced (Tiemann, 2007; Free Software Foundation, 2019). The Affero General Public License (AGPL) closed this 'loophole' by introducing a specific clause to include network distribution under the definition of distribution. In the server – web-application example, this would mean the server-code would need to be open-sourced to comply. In some licenses only changes to the component have to be disclosed. This is the case for the GNU Lesser General Public License (LGPL). In this case propriety software can include a LGPL licensed library and make use of the functionality provided. In case the library itself is modified, the license requires the modified library to be open-sourced.

2.1.2.9 | License Conflict with Sub-components

While an OSS component might display the license under which it is distributed, this license is not always valid. This is because OSS components might in turn use other components which are licensed different, or are not licensed at all. When these licenses are not compliant with each other there is a risk of theft of intellectual-property (Ruffin & Ebert, 2004; Sherlock et al., 2018). As licenses provide no warranty for the validity of the software in regards to the license the licensee is still responsible for any conflicts (Ruffin & Ebert, 2004).

2.1.3 | Benefits

While OSS components are suspect to a wide range of risks they do not come without benefits. Hauge et al. (2010) lists a wide range of potential benefits of using OSS components: shorter time-to-market; reduction in development, maintenance and license costs; community transparency regarding issues; and the quality, functionality and productivity benefits of software reuse. Since the majority of commercial software uses OSS components it can be concluded that OSS components are a requirement for most commercial viable software (Franch et al., 2013).

2.1.4 | Conclusion

The usage of OSS components includes qualitative risks as discussed in Section 2.1.2 and summarized in Table 2.2, which answers the question: *What risks are involved in using OSS components?* As can be seen in Table 2.2 the following risks are involved in using OSS components: security vulnerabilities, code integrity, reliability/quality, forking, changes in framework, lack of support, project discontinuation, license/copyright compliance and license conflict with sub-components.

Table 2.2: OSS risks related to their potential qualitative value impacts.

Risk	**Impact**
Security vulnerabilities	Damage to system integrity; Loss of data
Code integrity	Damage to system integrity; Loss of data
Reliability/Quality	Invalid functionality; Invalidation of data
Forking	Loss of support
Changes in framework	High update costs; Loss of support
Lack of support	Increased security, integrity and reliability risks
Project discontinuation	Loss of support
License/copyright compliance	Legal costs
License conflict with sub-components	Legal costs

2.2 | Software Valuation

All assets, including intangible assets such as software, have an intrinsic value. In order to valuate an asset the cash flow, discount rates and growth rates have to be taken into account. Based on these inputs the value of an asset for a company can be determined, which can determine the intrinsic value of the asset (Damodaran, 2013).

Discount rates of cash flows are based on risks. By looking into the risks of OSS components the discount rate on the integrating software can be estimated. This section will provide an answer to the question: *How can OSS component risks impact fair value?*

2.2.1 | Software as an Asset

Software is produced to provide use value. The use value is the value that the software provides in a form of economical benefits. This benefits exist for software after it has been put to use or has been sold (García García & Alonso Magdaleno, 2013). De Groot, Nugroho, Bäck, and Visser (2012) argue that software value can be looked at from three perspectives: exchange value, production value and use value. For this paper fair value will be used instead of exchange value as exchange value can be largely influenced by either an unwilling participant or a lack of transparency of information provision in a transaction. The fair value assumes an orderly transaction which means that value adjustments will be based on the product in a context were both parties are interested in a transaction of the product for the optimal price.

Fair Value "The price that would be received to sell an asset or paid to transfer a liability in an orderly transaction between market participants at the measurement date." (IFRS Foundation, 2018, p.671)

Production Value The cost to produce or replace the software.

Use Value The future economical benefits provided by the usage of software.

To determine the value for intangible goods, different valuation methods can be used. Some of these methods are as follow: the cost approach is based on the production cost of an asset which leads to the production value; the market approach bases the price of an intangible asset on similar assets available on the market, leading to the fair value; the income approach looks at the use value from a cash flow perspective, looking at the discounted cash flow with and without the component.

This paper will focus on the market approach to evaluate the value of software. Using this approach will lead to the fair value of software. Therefore, when the terms *value* or *software value* are mentioned it will be implied to mean the fair value of software. Different meanings of the word 'value' will be made explicit.

It can be difficult to determine which part of the cash flow is a result of the intangible asset. Holterman (2004, p.274–275) describes three methods to determine these cash flows: 'premium pricing', 'relief from royalty' and 'multi-period excess earnings'.

The premium pricing method compares a business with the intangible asset to a business without this asset. By comparing the profit and product pricing an estimate can be made for the cash flow impact of the intangible asset.

The relief from royalty method is used for intellectual property. When a piece of intellectual property is owned by a business no royalties have to be paid. Comparing the intangible asset to a comparable asset that has licensing costs available the value, or saved costs, can be determined.

The multi-period excess earnings method is the same as the earlier described income approach. This takes a cash flow of a business component from which the intangible asset is a component. By offsetting the cash flows that are generated by other assets the cash flow generated by the intangible asset can be determined.

Reilly (2008) states that the relief from royalty method can be used to determine the value of copyrighted software. In case the return on royalties is fairly related to the asset value, the relief from royalty method can also be used for a cost approach instead of a market approach.

When valuating software companies there is often the alternative of producing the software yourself. In a simple way this can be expressed as expecting a higher use value, minus the price paid, than the production value of the software. This

expression is limited though, as opportunity costs and the value of time-to-market should be incorporated as well.

2.2.2 | Cost of Risk

Section 2.1 talks about risks in OSS components. Risks can impact value as they can lead to future costs through damages, impact costs, or to earlier costs through mitigation effort. While impact costs are based on the uncertainty in risks, the probability of occurrence times the potential damages, mitigation effort is a proactive cost to reduce the probability of risk occurrences (Boehm, 2003). Linh et al. (2019) state that a mitigation strategy can be evaluated using the expected impact cost. The cost of mitigation should be lower than the reduction in the expected impact costs. In other words, the expected gain should be higher than the costs.

Nugroho, Visser, and Kuipers (2011, p.3–4) base Rebuild Value (RV), or effort to rebuild an application, on the System Size (SS) times a Technology Factor (TF) for the used technology. Multiplying this by the fraction, the percentage of Lines of Code (LoC) which have to be rewritten, gives an estimation of effort for change in person-months. According to Jones (1995); Nugroho et al. (2011) TF can be calculated using a technology productivity factor. A factor to convert LoC to effort based on the used technology. A later report by Jones (2017) provides the amount of effort per 1000 Lines of Code (KLoC). This data can be weighted by the percentage of code written in the relevant programming language to calculate TF. Using this data the mitigation effort can be calculated using the System Size (SS), the Technology Factor (TF) and the percentage of code which has to be rewritten to mitigate the risks, the Rebuild Factor (RF). Table 2.3 presents the methods to calculate these values. Table C.1 in Appendix C shows TF for relevant programming languages based on an effort in person-months.

De Groot et al. (2012) created three models for software as an asset on an enterprise level. These models are shown in their mathematical representation in Table 2.3. By taking the central ideas of these models they can be used to get a value adjustment for software as an asset.

The first model can be used to create an adjustment to the value based on the mitigation costs of the OSS component risks in the application. The second

Table 2.3: Mathematical representation of system and value variables.

Variable	**Formula**
System Size	$SS = LoC$
Rebuild Value	$RV = SS * TF$
Effort of Change/Mitigation Effort	$E(Ch) = RF * RV$
	$V_1 = V - MC$
Adjusted Value	$V_2 = V * (1 - RF)$
	$V_3 = V - \sum_{i=1}^{5} TI_i$

model ignores the effort that is required to change part of the system, but looks at the percentage of the system that needs change, RF. The value is adjusted according to this factor. The third and final model lays the focus on an increase in technical debt. Technical debt focuses on a compromise being made in software development to speed up product delivery (Ampatzoglou, Ampatzoglou, Avgeriou, & Chatzigeorgiou, 2015). For OSS components this could mean less time spent on proper adoption increasing the future risks of a component. Technical Interest (TI) is the increase in maintenance that comes from technical debt (Ampatzoglou et al., 2015; De Groot et al., 2012). TI extends to loss of productivity, defects and loss of quality (Avgeriou, Kruchten, Ozkaya, & Seaman, 2016). For OSS components it is relevant to include the potential risk damages under defects which can be caused by the technical debt of using at risk components.

The final model assumes not resolving technical debt and paying the price on Technical Interest (TI) instead. The value is adjusted to the cost of TI over an amortization time of five years. TI can be difficult to quantify as it is an estimate on future cost. At the same time it is possible that there are no costs over technical debt, meaning that TI is zero (Ampatzoglou et al., 2015). It can be hypothesized that a properly adopted OSS component only has technical debt based on the risks shown in Table 2.1, as there is no debt from adoption risks. This would mean that TI equals the expected impact cost of the risks found in OSS component usage.

2.2.3 | Quantifying Risk Costs

The expected impact costs of risks in OSS components can be based on the qualitative damages shown in Table 2.2.

The global average cost of a data breach is $3.92 million, with a cost of $150 per lost record (Ponemon Institute, 2019, p.3). These costs are relatively higher for smaller companies versus bigger companies, expressed in cost per employee. Mitigation strategies such as data encryption can reduce breach costs on average by $360.000 (Ponemon Institute, 2019, p.8).

The legal costs for "big" companies can be around $300.000 per month in the United States (Rosen, 2004, p.271). When looking at statutory damage costs for copyright infringement the costs range from $750 to $30.000, while willful infringement tops at $150.000 (Rosen, 2004, p.273).

Software Freedom Conservancy v. Best Buy Co. (2010a) shows that the statutory damages are uphold in court for violations of open source licenses, which in this case was a violation of a GPL license. In addition to those costs, legal fees for the opposing party can be claimed as is shown in Software Freedom Conservancy v. Best Buy Co. (2010b). Further injunctive relief[1] can be required which can further impact business until the violation is resolved, which in this case meant the direct prohibition to further distribute hardware that included copyrighted software (Software Freedom Conservancy v. Best Buy Co., 2010a).

All open source licenses protect the licensor from any and all liabilities. In case of conflicts between sub-components the licensee would be responsible at all times. According to Rosen (2004) in different countries consumer right protections might invalidate this disclaimer. This means that copyright infringement due to license conflict can be lower as partial responsible goes to the licensor. This would only apply in cases where the licensor can be held at least partially responsible for conflicts. In the case of Welte v. Fantec (2013) Fantec argued for lesser responsible because their contractor would be liable for the infringement of the GPL license. In this case it was ruled that the distributor, Fantec, was itself responsible for licenses in its sold product.

In case there is no more community support for an OSS component the imple-

[1] "a court-ordered act or prohibition against an act or condition" (The Free Dictionary, n.d.)

menter is responsible for maintenance when issues arise. The effort for maintenance can be quantified according to the model of Nugroho et al. (2011, p.5) which states that maintenance effort is based on the quality of a system in combination with the fraction which needs active maintenance.

Based on the qualitative impacts of the risks shown in Table 2.2 a categorization can be made to match the qualitative impacts to their quantitative counterparts. Risks where the quantitative impact is expected to be similar are collapsed under a single category as seen in Table 2.4.

Table 2.4: Qualitative literature risks categorized in quantitative categories.

Literature Risks	**Quantitative Category**
Security vulnerabilities Code integrity Lack of Support	Security risks
Reliability/Quality	Risk of failure
Forking Project discontinuation Changes in framework	Loss of support
License/copyright compliance License conflict with sub-components	Copyright infringement

2.2.4 | Conclusion

The impact of risks in OSS components on the fair value of software can be quantified in three different ways: by subtracting the mitigation cost of the risks, by deducting the percentage of the software that needs to be refactored to mitigate the risks or to deduct the damage of the risks over a five year period.

Section 2.2.3 quantifies the damages of security risks, loss of support and copyright infringement. These damages are summarized in Table 2.5. The damages shown in Table 2.5 can be adjusted based on mitigation strategies, such as data encryption, and might differ between countries. These damages will be used as a baseline for the cost of risk damages of the risks in OSS components.

Table 2.5: Potential damages of risks in OSS components.

Risk Category	Risk Damages
Security Risks	\$150 per lost record or \$3.92m total on average (global)
Loss of Support	Maintenance Effort
Copyright Infringement	\$750–\$150.000 (US) + Legal costs (\$300.000 monthly (US)) + Compliance costs

3

Risk Model Design

This chapter will explain the methodology to estimate the downward impacts of risks on software value. As stated in Section 1.2 the objective of this research is to create a model that can estimate the risk impacts on the value of software. To achieve this, initial risk models will be created based on the literature explored in Section 2. Due to the limited amount of data available involving the relation between OSS risks and software value a set of case studies will be used to research this relationship. These case studies will include a risk analysis for OSS components in software based on the developed risk models. The impact of these risks on the value of software will be researched in the case studies, which are outlined in Section 3.5.

3.1 | Framework Selection

Different frameworks are proposed in research to model software systems. The relevance of such frameworks depends on the areas of these systems which are to be analyzed. As this research is focused on the risks in OSS components, proposed frameworks should enable the modelling of risk properties which can be used to identify these risks. The identification of risks can be done through obstacle analysis, in which risks are looked at from a goal perspective. The non-satisfaction of these goals can be measured through obstacles, risks, which satisfaction determines the non-satisfaction of the above goals through a bottom-up approach (Cailliau &

van Lamsweerde, 2013).

Yu and Mylopoulos (1994) introduce a goal-oriented modelling framework called *i**. In this framework goals are subdivided in sub-goals. Each goal can positively or negatively relate to other goals, providing a top-down approach for modelling systems. Goals are identifiable by measurable tasks and affected actors. This helps in identifying different paths towards a goal and the related actors in these paths.

Different goal-oriented models have been proposed originating from the *i** framework with different focus areas. Sebastiani, Giorgini, and Mylopoulos (2004) propose a framework based on a minimum-cost goal approach, Asnar, Giorgini, and Mylopoulos (2011) look at risks from a requirement engineering perspective, while Siena, Morandini, and Susi (2014) base their framework around OSS component risks. In the framework of Siena et al. (2014) goals are sub-divided in events which cause non-satisfaction in the goals similarly to the obstacle analysis of Cailliau and van Lamsweerde (2013). These events have a likelihood and severity of their occurrence, which combined negatively affects the goals. These values can be conceptually represented through situations which are measurable through indicators.

As the framework by Siena et al. (2014) is explicitly developed for the modelling of OSS component risks this framework enables us to show the relation between risks of OSS components and their identifying properties.

3.2 | Initial Model Design

Siena et al. (2014) describe the formal definition of the RiskML modelling language. Based on literature research into the risks of OSS components initial risk models are created as can be seen in Figure 3.1 and Figure 3.2. The graphical representation of the models is created using Microsoft Visio (Microsoft, 2019). The aim of these model is to create a graphical representation of the risk categories shown in Table 2.4. By using the RiskML modelling language OSS risks can be modelled in a top-down approach from negatively impacted goals down to their indicators. A negative impact on these goals, as represented by events, will reduce the value of software. This value reduction can be represented through quantified methods and costs as described in Section 2.2.2 and 2.2.3. It is important to note that RiskML

only provides a negative connection between events and goals as defined by the 'Impact' relation.

As described by Siena et al. (2014) RiskML models are built up from different concepts and relations:

- **Indicator:** Representation of a measurable property which is evidence for the existence of a situation. An indicator can be used as a measurement of the satisfaction, the measurement of truth, of a situation. This relation is modeled using the 'Indicate' relation, where the indicator satisfies the situation as being true.

- **Situation:** Situations present circumstances under which risk events can occur. The existence of these circumstances can be measured through the indicators of the situation. How situations relate to risk events are modelled through the 'Expose', 'Protect', 'Increase' and 'Mitigate' relations. The increases or decreases from these relations are relevant relative to the satisfaction of the situation.

  - **Expose:** Increases the likelihood of a risk event occurrence.
  - **Protect:** Decreases the likelihood of a risk event occurrence.
  - **Increase:** Increases the severity of the consequences of a risk event occurrence.
  - **Mitigate:** Decreases the severity of the consequences of a risk event occurrence.

- **Event:** Events represent possible occurrences which have a negative impact on goals through the 'Impact' relation. These concepts have a likelihood and severity property which are measured through situations.

- **Risk:** The model as a whole represents the uncertainty and consequences of a risk.

The relations between the different concepts are summarized in Table 3.1.

Table 3.1: Simplified propagation rules of RiskML models adopted from Siena et al. (2014, p.8).

Relation	Influence	Type	From	To
Indicate	Increase	Satisfaction	Indicator	Situation
Impact	Decrease	Satisfaction	Event	Goal
Expose	Increase	Likelihood	Situation/Event	Event
Protect	Decrease	Likelihood	Situation/Event	Event
Increase	Increase	Severity	Situation/Event	Event
Mitigate	Decrease	Severity	Situation/Event	Event

3.2.1 | Security Risks

Coelho, Valente, Silva, and Shihab (2018) researched the indicators to measure levels of maintenance for OSS projects on GitHub. The created model fits projects into categories of being maintained and not being maintained. Projects that are classified as finished are also considered unmaintained, even though these might still solve reported issues. According to Coelho et al. (2018, p.4) the most important indicators are the following:

- Amount of commits ($T_{22,24}$)
- Maximum amount of days without a commit ($T_{22,24}$)
- Maximum days without commits ($T_{10,12}$)
- Maximum contributions by developer ($T_{16,18}$)
- Amount of closed issues ($T_{1,3}$)

Indicators are most relevant within a time interval of months. This is shown by their T value, meaning a T value of 22,24 means the indicator applied over the last 22 to 24 months (i.e. amount of commits in the last 22 to 24 months). These indicators depict the situations of *Low activeness*, *Lack of active contributors* and *Issues are actively resolved* as can be seen in Figure 3.1. The amount of contributions by developers is a quantification of the lack of support coming forth through forking. A lack of active contributors means that the impact of a developer leaving the

project has a bigger negative impact on the support level of the project than what would be the case with more active contributors.

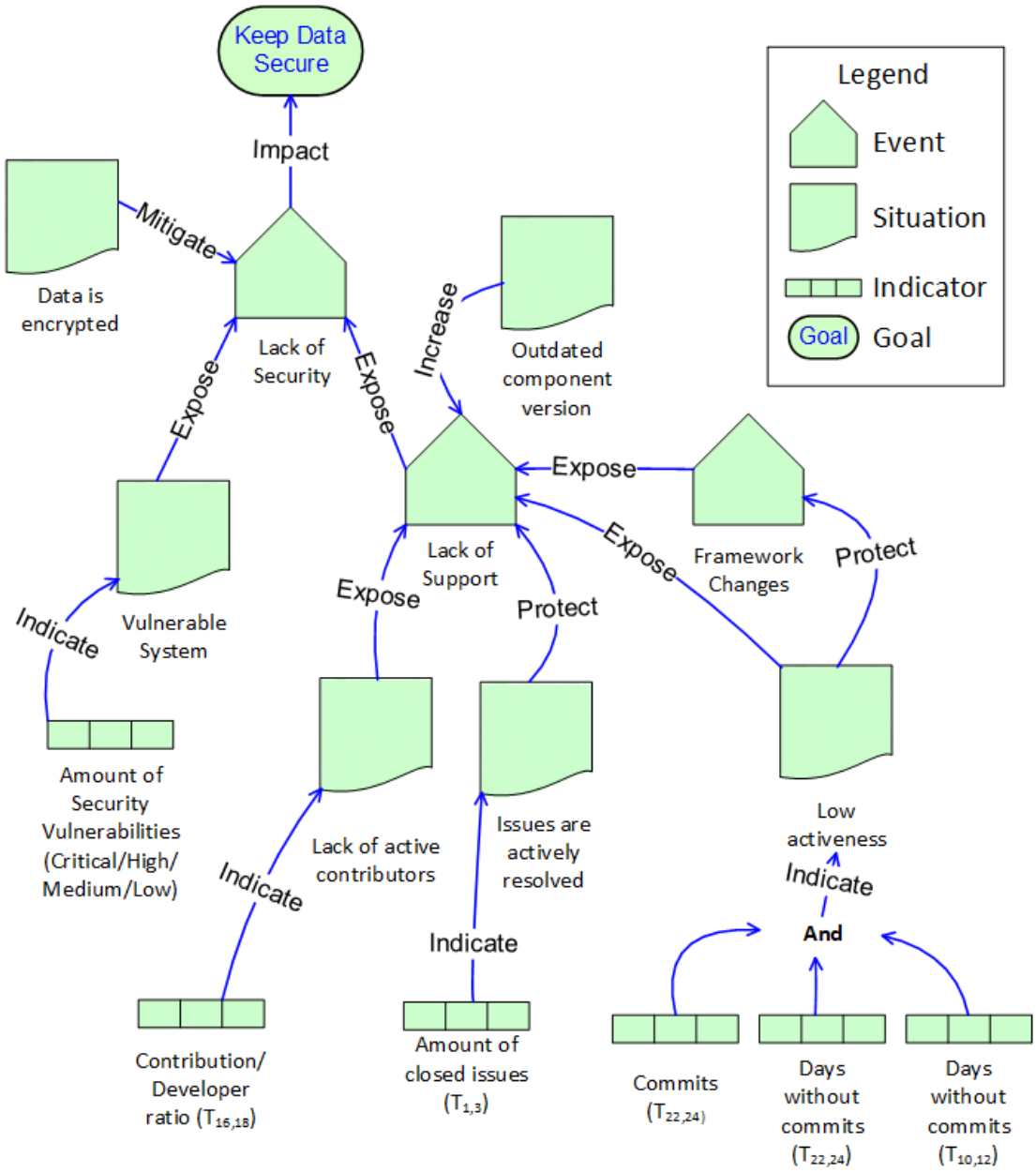


Figure 3.1: Security Risks model based on literature.

Vulnerabilities are categorized from low to critical as is equal to the way CAST Highlight (CAST, 2020b) categorizes security vulnerabilities in OSS. Lowering

the damage of security issues can be done by taking data security measurements as described in Section 2.2.3. The interaction between activeness and framework changes is based on the RiskML model by Siena et al. (2014).

3.2.2 | Copyright Compliance

Copyright compliance in OSS components is done by compliance with the license provided with the component. Further compliance has to be done with all code and components used in the licensed component. Depending on the license, compliance might require open-sourcing propriety code, but such requirements might also depend on the means of distribution (Tiemann, 2007; Free Software Foundation, 2019). The combination of the distribution situation and the type of copyleft, which describes what part of the code needs be distributed under the same license, exposes the non-compliance event as seen in Figure 3.2. Compliance with sub-components and code used in the component is in general automatically satisfied when there is compliance with the license. The exception here is license conflicts which causes non-compliance, as discussed in Section 2.1.2.9 and described by Sherlock et al. (2018); Ruffin and Ebert (2004). This means that licenses try to cover a part of the code used in the component which they can not rightfully license, thus their license does not actually cover this part of the code. As licenses explicitly state that they carry no liability in any form, the licensee of the component would be responsible for this issue (Rosen, 2004). If this is uphold, which depends on the country, the implementer would be responsible for damages (Rosen, 2004; Ruffin & Ebert, 2004). The *Component has conflicting licenses* in Figure 3.2 shows that this exposes the risk of non-compliance. As further discussed by Rosen (2004) the difference in risk impact exists in willful versus non-willful copyright infringement. This is further exemplified in Section 2.2.3 by, among others, the Software Freedom Conservancy v. Best Buy Co. (2010a) case. In Figure 3.2 this is shown by the increase relationship of the *license is ignored* situation to the non-compliance event. As Sherlock et al. (2018) state, it is best to have gather legal council on the matter to assure proper compliance. This means that the event of *Non-compliance* can be protected by gathering legal advice on the matter.

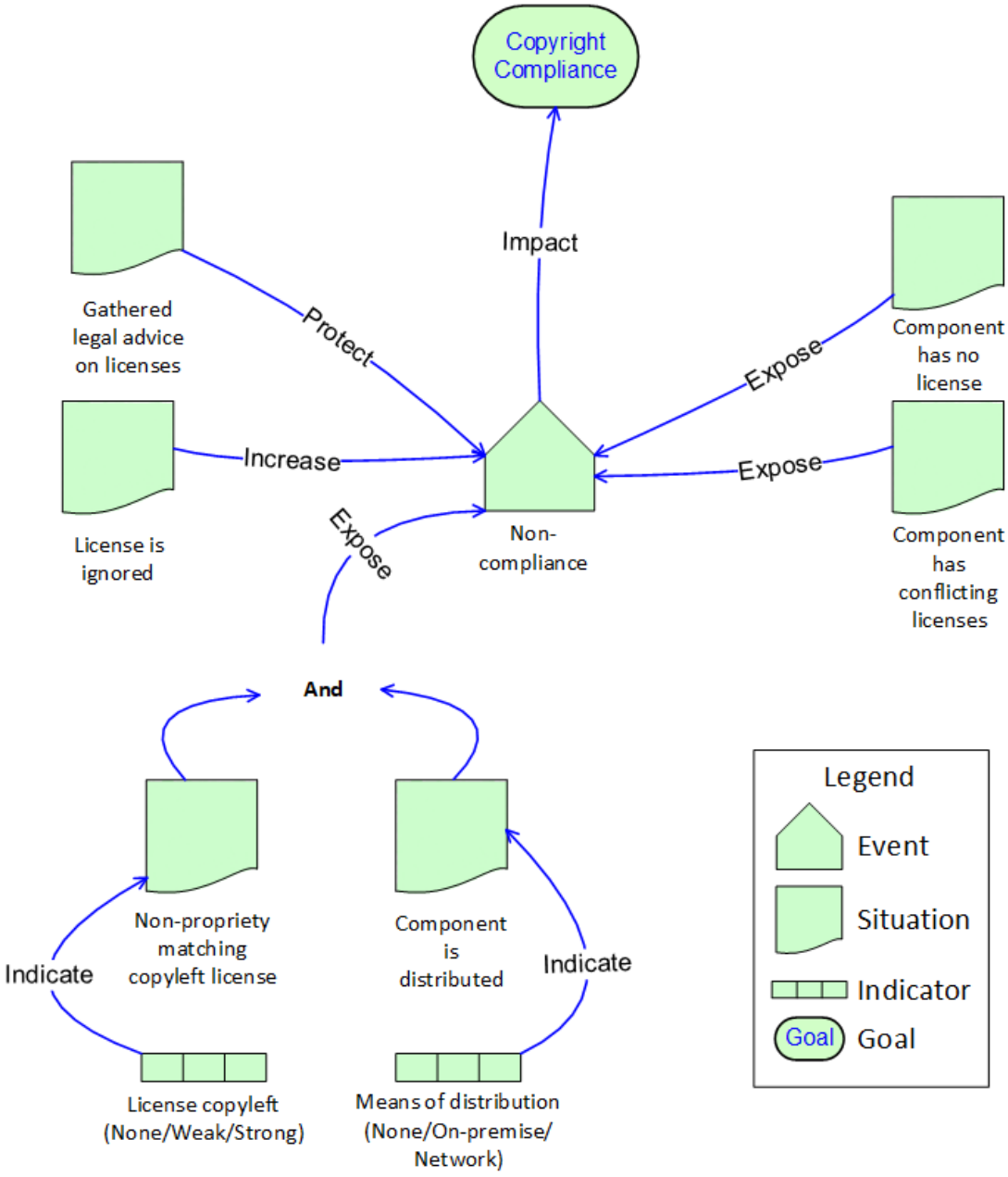


Figure 3.2: License Risks model based on literature.

3.3 | Model Implementation

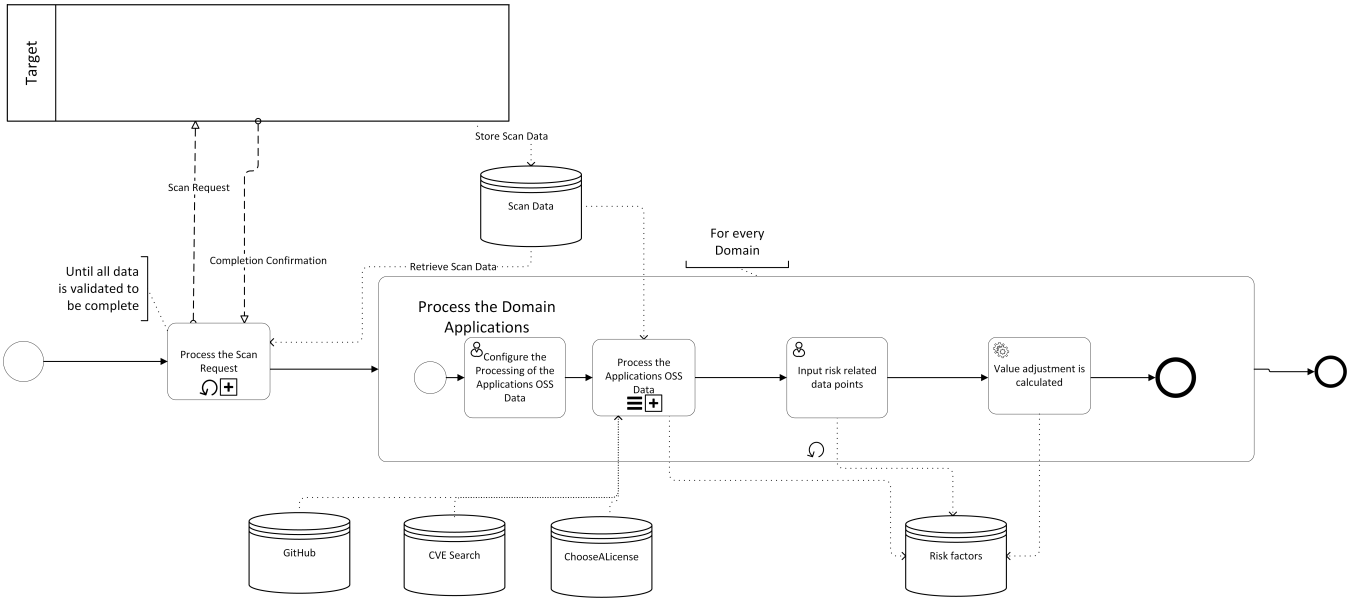


Figure 3.3: Business Process Model and Notation (BPMN) of the implementation of the risk models. Expansions of the sub-processes are shown in Figure 3.5.

The graphical models displayed in Figure 3.1 and 3.2 will be implemented using R: A Language and Environment for Statistical Computing (R Core Team, 2020). The models will use OSS lists provided by CAST Highlight (CAST, 2020b). Based on the version, programming language and name of the OSS component the correct repository can be looked up on GitHub. Choosealicense.com (2019a) and CIRCL (2020) will be used to look up more information regarding the risks of the components.

The model in Figure 3.3 depicts the business flow of using the model implementation of the models shown in Figure 3.1 and 3.2. The model in Figure 3.3 was created using Microsoft Visio (Microsoft, 2019) and is modelled according to the BPMN 2.0 guidelines. A minimal legend of the used BPMN symbols in these figures can be seen in Figure 3.4.

The first step in the business process is to acquire a CAST Highlight (CAST, 2020b) scan on the applications of the target of the transaction, the party that is targeted to be acquired. The CAST Highlight (CAST, 2020b) scan software is

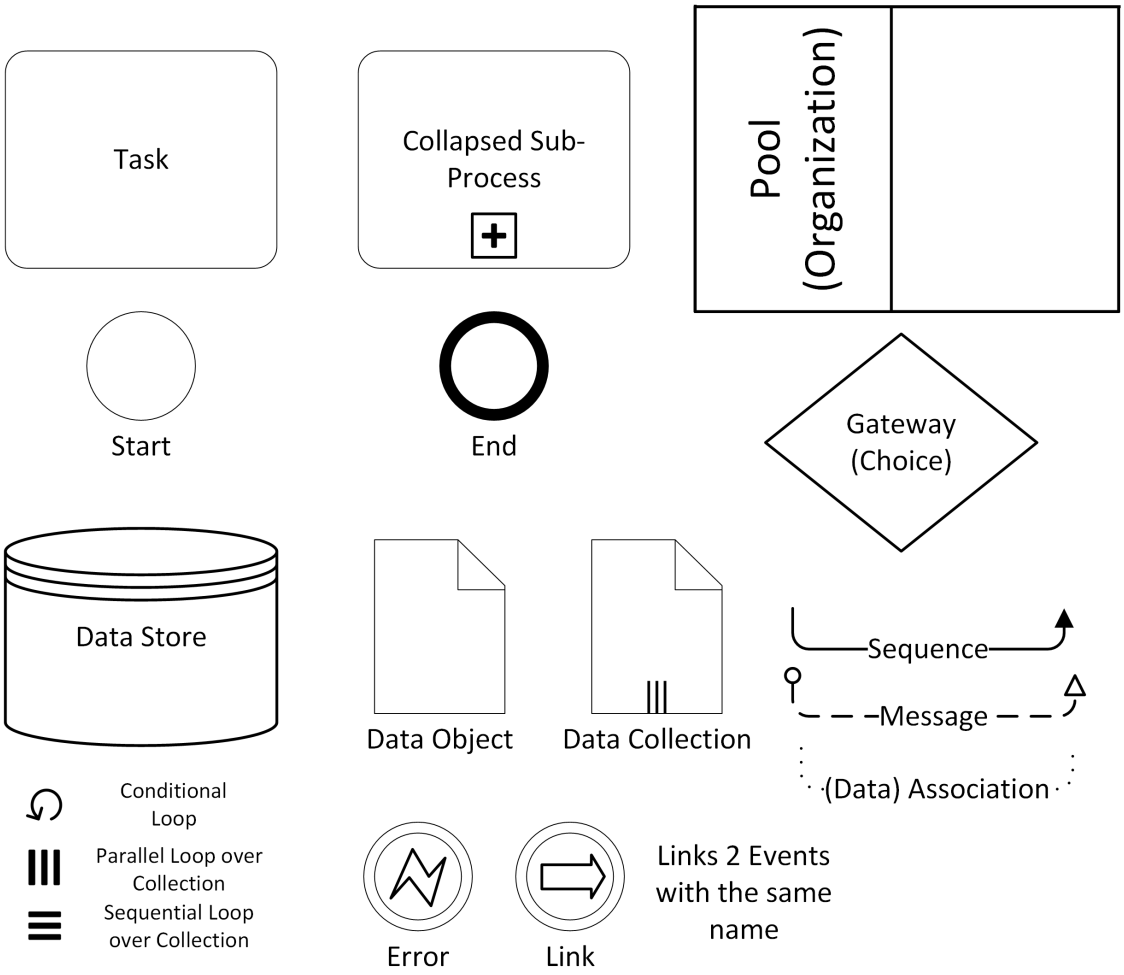
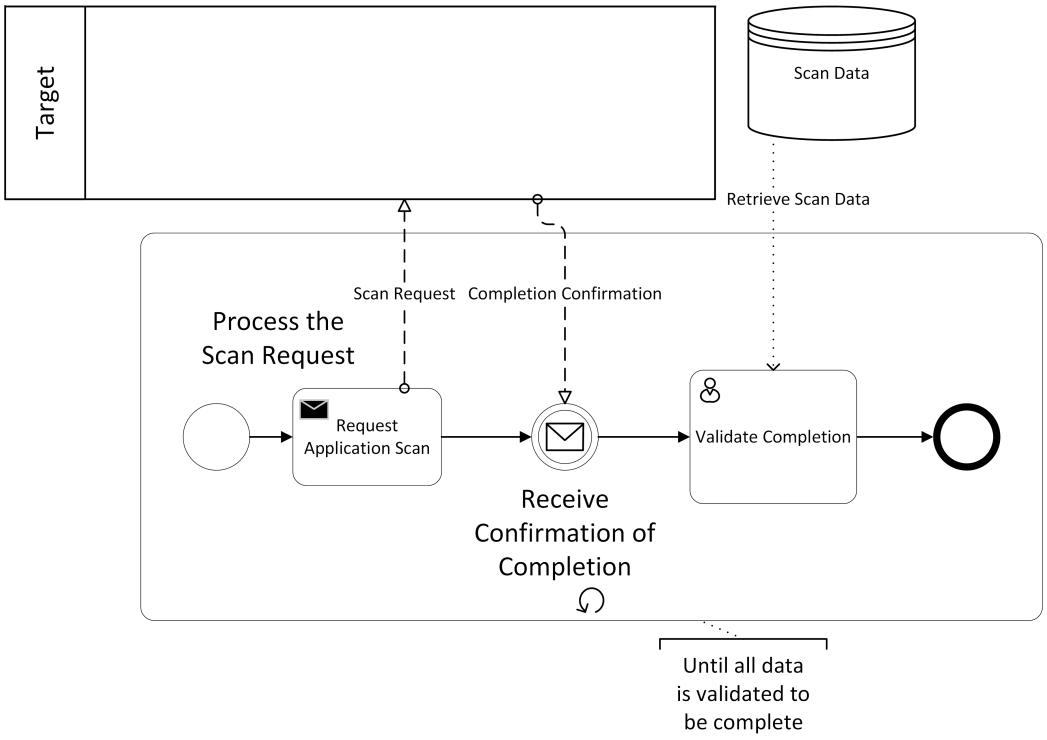


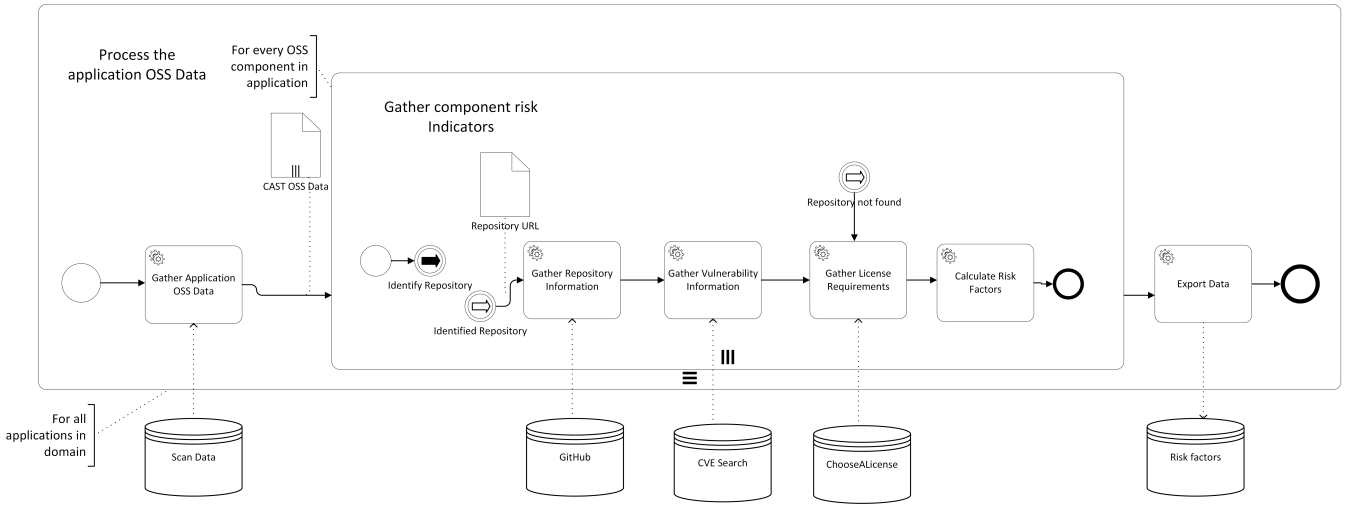
Figure 3.4: BPMN symbols used in this thesis.

used to scan the code of the applications, and submits these results to a database. This can be seen in Figure 3.3 by the association between the *Target* and the *Scan Data*.

Figure 3.5a shows the sub-process of the scan request. The scan is requested by EY which includes instructions on how to perform the scan. After the target completes the scan, a confirmation will be sent to EY. A manual check of completion is done against the scan data, this is done by validating the scanned results against the known specifications of the applications. When it is determined that the scan is missing certain applications the sub-process is repeated, otherwise the



(a) Process Scan Request sub-process.



(b) Process Application OSS Data sub-process. Linked identify repository process part is shown in Figure 3.6.

Figure 3.5: Expanded sub-processes of the business process model.

scan data can be used and analyzed.

CAST Highlight (CAST, 2020b) provides an overview of different code metrics based on the results from the scan. This business process will focus exclusively on the OSS scan results. As can be seen in Table 3.2, CAST Highlight (CAST, 2020b) provides different data points about the used OSS components in the scanned applications. The name, programming language, and version will be used to look up the repository on GitHub. The vulnerabilities and license data points are later used for the risk analysis.

The expanded sub-process shown in Figure 3.5b depicts the processing of the application OSS data. This is done as a batch for every application in a domain; a category of applications from the target. This process is then repeated for every domain that has to be analyzed. A manual configuration is required for every domain, *Configure the Processing of the Applications OSS Data* in Figure 3.3, after which the automated process, *Process the Applications OSS Data*, can be started.

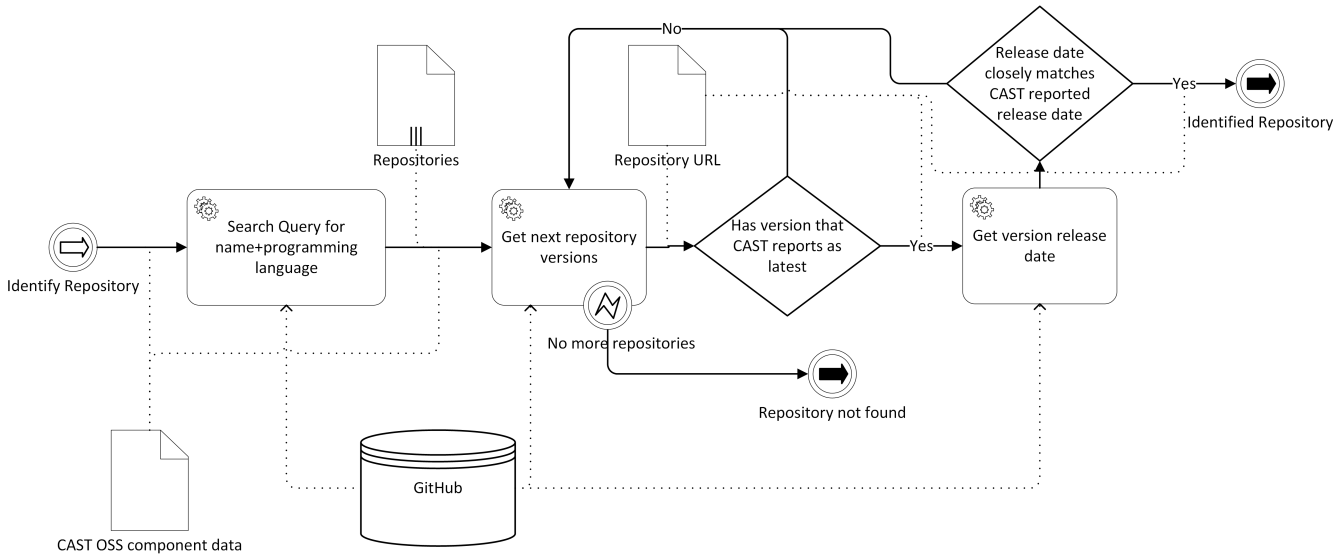


Figure 3.6: Linked identify repository process part of Process Application OSS Data sub-process (Figure 3.5b).

The first automated step is to retrieve the OSS data from the CAST Highlight (CAST, 2020b) scan. This data is required to find the correct repository on GitHub. This is done by searching GitHub for repositories with matching names and written in the programming language identified by CAST Highlight (CAST, 2020b). As

can be seen in Figure 3.6 the correct repository can be identified by going through the list of matching repositories on GitHub and matching the version and release date to the ones reported by CAST Highlight (CAST, 2020b).

When the correct repository is identified, data from the repository can be gathered; see *Gather Repository Information* in Figure 3.5b. As shown in Table 3.2 the following data points are retrieved and calculated from GitHub: *Amount of commits*, *Days without commits*, *Amount of closed issues* and *Contribution/developer ratio*. These data points are used as indicators for the *Lack of active contributors*, *Issues are actively resolved* and *Low activeness* situations in the risk model shown in Figure 3.1. The *Component versioning* data point is also collected which can directly satisfy the *Outdated component version* situation.

The *Gather Vulnerability Information* task can satisfy the *Amount of Security Vulnerabilities* in the Security Risks model (Figure 3.1). This indicates the satisfaction of the *Vulnerable System* situation. Vulnerabilities are provided in the CAST Highlight (CAST, 2020b) output but can also be looked up using CVE Search (CIRCL, 2020). This can provide an overview for vulnerabilities in the used version of the component as well as the vulnerabilities in the latest version of the component. By comparing the vulnerabilities between versions it becomes clear whether there is a possible way to mitigate the vulnerabilities by upgrading the component.

The next tasks in the sub-process shown in Figure 3.5b can further satisfy different indicators and situations in the risk models. *Gather Copyleft Information* can satisfy the *License copyleft* indicator in the Copyright Compliance model (Figure 3.2). This is done by collecting the requirements of a license from Choosealicense.com (2019a), which data is shown in Appendix A. The *Means of distribution* as indicator of *Component is distributed* is collected as manual input. The satisfaction op the situation *Component is distributed* combined with the situation *Non-propriety matching copyleft license* can expose the *Non-compliance* event. This task is also executed when the repository is not found on GitHub. In this case the *Scan Data* will be directly used to determine the component license, without the validation of GitHub. This non-validation will be included

By gathering all available risk data in the *Gather component risk Indicators* sub-process, shown in Figure 3.5b, the likelihood of the event *Lack of Support*

occurrence can already be calculated (Figure 3.1). With the satisfaction of the situations *Low activeness*, *Issues are actively resolved* and *Lack of active contributors* and the Random Forest implementation of Coelho et al. (2018) an estimate can be made on whether the support of the component will last. The method of the Random Forest implementation is further described in Section 3.2.1.

Finally in the final step of the *Process application OSS Data* sub-process, all collected data will be exported to an Excel readable file. The data in this file can be used in estimations of fair value adjustments, based on the calculated risk factors and mitigation strategies.

Table 3.2: Required data points and their sources for the model implementation.

Source/Data Points	**Manual Input**	**CAST Highlight (CAST, 2020b)**	**GitHub**	**CIRCL (2020)**	**Choosealicense.com (2019a)**
Distribution form	✓				
External risk protection factors	✓				
Component name		✓			
Component programming language		✓			
Component versioning		✓	✓		
Security vulnerabilities		✓		✓	
License		✓	✓		
License copyleft					✓
Amount of commits			✓		
Days without commits			✓		
Amount of closed issues			✓		
Contribution/developer ratio			✓		

3.4 | Model Validation

For every case a set of data will be collected. These account for the following data points:

- Which applications are part of the case
- Types of OSS risk findings (Security, Loss of Support, Copyright infringement)
- CAST Highlight (CAST, 2020b) output for application if available
- Cost analysis for OSS risks if available
- Application rebuild values
- Application sizes in LoC
- Major programming language(s) used
- Relative size of application written in the major programming language
- Scope of the engagement:
    1. Software Quality Assessment
    2. Software Composition Assessment
    3. Software Sustainability Assessment
    4. Software Due Diligence (SDD)
- Industry sector according to the broad Statistical Classification of Economic Activities in the European Community (NACE) sections (European Commission, 2008, p.57)

These will be used in the case studies to assess the risks in the applications and create an estimated range of costs to mitigate these risks. The scope of an engagement is categorized in four types of assessment sorted by the depth of the assessment. A Software Quality Assessment is the most basic form and SDD is the most inclusive engagement form.

To better group the different industries these will be categorized according to NACE. This provides an existing categorization according to the European standard. The cases will be associated with the sector they provide software for. As the amount of cases is limited only the top level categories will be used which are shown in Appendix B.

Finally, cost analyses will be gathered to create an understanding of value adjustments based on OSS risks. This will help in creating an approach for quantification of these risks.

The risks of the components in the case applications will be assessed based on the models shown in Figure 3.1 and 3.2. The following data points will be collected in the unstructured interviews related to the models:

- Missing indicators, situations, events and/or goals.
- Incorrect indicators, situations, events and/or goals.
- Incorrect or missing relations.
- General feedback on the model

The conducted interviews will help in validating the created models. Furthermore, these interviews will help in creating a more complete picture of how risks can be measured through missing or incorrect situations and indicators. Based on this data an iteration of the models shown in Section 3.2 will be created. Furthermore, case data will be analyzed through the model implementation in the case studies in Section 4.4. In this section financial data will be used to create an understanding of fair value adjustment based on risks in OSS components.

3.5 | Outline of Case Studies

Past merger and acquisitions of EY will be used as case studies to look into risks of OSS components. Cases were selected based on the availability of OSS data. Engagements were any aspect of OSS analysis was included will be used for this research. Based on this, three cases were identified.

3.5.1 | Case Study Approach

For every case OSS component data will be collected using the tool developed as described in Section 3.3. Based on the output, the understanding of how risk assessment can be used in the value determination of software will be improved and validated.

Mitigation effort ranges will be based on data from Case A as this is the only case which provides value estimations. This case will be used to determine minimum and maximum effort for replacement and update mitigation strategies. These ranges will be applied in all cases to validate our approach to value determination. Because of this, this will be the first case to be analyzed.

Based on the rebuild value and the estimated effort to mitigate risks, effort ranges for each component are created. According to Halkjelsvik and Jørgensen (2018) time estimations with a minimum and a maximum value, aggregated by their median value, do not give an accurate representation of actual time required. One solution for this is a three point estimation according to the Program Evaluation and Review Technique (PERT), although this assumes an estimation accuracy of 99% for the maximum and minimum value. Halkjelsvik and Jørgensen (2018) state that this accuracy is often not achieved. To get an accurate time prediction, historical time predictions and their accuracies can be used to create a time distribution. Using this data an accurate prediction can be made using an adjusted mean value of the time ranges.

As historical data of is not available, it is not possible to directly apply the method of Halkjelsvik and Jørgensen (2018). Instead we apply the Monte Carlo method to try to achieve similar results. This enables us to create an effort distribution even though the available data is limited. In Section 4.4.1.1 this approach will be further described.

Every case study will be divided in two parts. One part will discuss general observations which will also include the output of the Static Analyzer and the recommendation of the Risk Analyzer from the developed tool. The second part will discuss the value determination based on the described values and approach to value determination in Case A. This aligns with the Fair Value Adjustments component of the Risk Analyzer. Case A will contain an extra part, Understanding

Value, which will create an understanding of value based on the data in this case.

3.5.2 | Case A

Case A consists of a full SDD. This is the only engagement that includes a financial assessment of the costs for OSS component risks. These financial assessments will be used to create an understanding to the question: *How can the costs of mitigating OSS component risks be quantified?* This understanding will be applied in and validated through all case studies. There are 16 applications for which CAST Highlight (CAST, 2020b) outputs are available for license and security risks.

3.5.3 | Case B

Case B consists of a Software Sustainability Assessment for a target in the 'transportation and storage' sector. This case contains 18 applications for which a CAST Highlight (CAST, 2020b) analysis is available. The analysis contains license and security risks found in the used OSS components.

3.5.4 | Case C

The sector of Case C is software for the 'human health and social work activities'. The scope of the engagement consisted of a Software Sustainability Assessment. 30 applications are part of this engagement. These are available as CAST Highlight (CAST, 2020b) output. This engagement does not include a cost quantification for the OSS component risks.

4

Model Validation

4.1 | Expert Validation

The models in Figure 3.1 and 3.2 have been validated using unstructured interviews with experts on the topic. Results of these interviews will be outlined in this section. Findings of the interviews represent inaccuracies, incompleteness or general comments regarding the models and their representation of the risks. Based on these interviews adaptions have been made to the models which will be shown in this section. The first iteration of these models have been separately validated by unstructured interviews with different members of the software team for which the results are shown in Section 4.1.1.2 and 4.1.2.2.

4.1.1 | Security Risks Model Findings

Table 4.1, 4.2 and 4.3 show the results of the interviews about the model shown in Figure 3.1. These results were used to improve the models accuracy and completeness. The first iteration, which was made by combining the results of the first three interviews, was used in the last two interviews. Table 4.4 and 4.5 show the results of these last two interview.

4.1.1.1 | Original Model Findings

Table 4.1: Results of the first interview of the Security Risks model validation.

Category	Findings
Missing information	Issue size of closed issues might be relevant to include. Vulnerable system could use more indicators to predict system vulnerability based on past vulnerabilities. The amount of breaking changes between versions can indicate how much support there is for older versions. Encryption is one way to mitigate/protect the Lack of Security, more methods exist.
Incorrect information	Low activeness should be changed to Low project activity. Amount of closed issues should be related to the ratio of issues. Framework changes might cause an outdated component, not a general lack of support. Low activeness does protect framework changes, although this is not relevant as it does not indirectly protect Lack of Support.
General Feedback	Ensure readability for the model to improve the adoption of the model.

Table 4.2: Results of the second interview of the Security Risks model validation.

Category	Findings
Missing information	Protection through separation of OSS components and data (no touch). Active support of big companies through monetization programs increases the trust in the component and with that increases the usage and support. Business processes to ensure updated libraries. IT processes to ensure system security (Firewalls, Automated penetration tests, ISO 27001 compliance).
Incorrect information	Outdated component create a direct lack of security, this is less or not related to the level of support. Commits indicators should together form one 'decline in commit activity' indicator.
General Feedback	Vulnerability prediction (building upon the Vulnerable System situations) might give a better overview of the security but this should be outside the scope of this model. Different colors between indicators, situations, events and goals for clarity. Structure the model in layers of indicators, situations, events and goal for clarity.

Table 4.3: Results of the third interview of the Security Risks model validation.

Category	Findings
Missing information	OSS policies can mitigate the impact of framework changes.
Incorrect information	Issues should be an indicator of the ratio between opened and closed. Framework changes will only expose lack of support when there are no key contributors. At the same time when there are no active contributors, such changes are unlikely to happen. One security vulnerability is enough to make a system vulnerable. The amount is less, or maybe not, relevant.
General Feedback	Add the arrow (relationship) to the legend Radical framework changes, as shown in the model, are very unlikely to happen.

4.1.1.2 | Iterated Model Findings

Table 4.4: Results of the fourth interview of the Security Risks model validation. These results are based on the initial iteration of the model.

Category	Findings
Missing information	Indicator for outdated component. This is a measurable situation. System security has more goals than 'Keep Data Secure': malicious code injection, hostile system takeover, acting as the system for malicious purposes. These also cause business damages.
Incorrect information	
General Feedback	

Table 4.5: Results of the fifth interview of the Security Risks model validation. These results are based on the initial iteration of the model.

Category	Findings
Missing information	Possible ways to measure the vulnerability of a system would be to look at amount of bugs/issues reported and how many are resolved. If too many (above a certain threshold) issues are reported this might indicate that there might exist more issues, unknowingly, in the system. Automated tests for components, instead of just for the system, might protect a component from being vulnerable.
Incorrect information	
General Feedback	System security and data security can be manually measured by a type of score sheet, indicating how 'good' a company is at security processes and policies.

4.1.2 | Copyright Compliance Model Findings

The results of the validation of the Copyright Compliance model, shown in Figure 3.2, are shown in Table 4.6, 4.7 and 4.8. Based on these results the model

was improved for accuracy and completeness, and further validated with two more interviews. The results of the last interviews are shown in Table 4.9 and 4.10.

4.1.2.1 | Original Model Findings

Table 4.6: Results of the first interview of the Copyright Compliance model validation.

Category	Findings
Missing information	Company policies for licenses. The copyleft and distribution situations require a third situation for compliance. This is the method of usage; whether it is used in a modified form, used as an 'as-is' code implementation or used as a binary. The requirements of different licenses are not included to expose the non-compliance event. Licenses can require disclaimers, a public code base and more.
Incorrect information	
General Feedback	

Table 4.7: Results of the second interview of the Copyright Compliance model validation.

Category	Findings
Missing information	The deliberate removal of a license and acting like its ones own code. CI/CD license compliance checker. Business processes to ensure compliance. Company policies for licenses.
Incorrect information	
General Feedback	Improve clarity (as described in Table 4.2 under General Feedback)

Table 4.8: Results of the third interview of the Copyright Compliance model validation.

Category	Findings
Missing information	Some components can have a dual license, this is missing in the model. Periodic OSS compliance checks can help a system to improve their compliance with licenses.
Incorrect information	"None" should be probably not be part of the copyleft and distribution indicators, as that would assume the indicator is satisfied when there is no copyleft or distribution.
General Feedback	

4.1.2.2 | Iterated Model Findings

Table 4.9: Results of the fourth interview of the Copyright Compliance model validation. These results are based on the initial iteration of the model.

Category	Findings
Missing information	Add indicators for situations that are measurable through automated data collection.
Incorrect information	
General Feedback	Switch around 'Business OSS policies are in place' and 'License specific requirements are not met' for clarity.

Table 4.10: Results of the fifth interview of the Copyright Compliance model validation. These results are based on the initial iteration of the model.

Category	Findings
Missing information	Components with dual licenses might be exposed/not-exposed to non-compliance because of one license or the other.
Incorrect information	
General Feedback	

4.2 | Model Iterations

Iterations of the risk models were created based on the results of the unstructured interviews. The first iterations of the models were created based on the results of the first three interviews of which the results are shown in Section 4.1.1 and 4.1.2. These iterations were used in the last two interviews to be further validated and completed. The final iterations are shown in Figure 4.1 and 4.2. In these iterations changes are colored. In the case of an added symbol the symbol is outlined in orange, if a relation is added or its end-point has changed it is also colored. Finally if a symbol is not added, but the text is changed, only the text is colored to show change.

4.2.1 | Security Risks model Iteration

Figure 4.1 shows the final iteration of the Security Risks model. This iteration is based on the results of the interviews as shown in Section 4.1.1. Results shown in Table 4.1, 4.2 and 4.3 are based on the model shown in Figure 3.1. The results shown in Table 4.4 and 4.5 are based on an intermediate iteration. The final iteration is an extension of the intermediate iteration based on the results of the final two interviews.

The results of the initial interview show that the initial model missed situations around OSS policies and security measurements. Table 4.2 mentions examples of IT process that can help ensure system security. These security measurements are collected in a single new situation *IT Security systems and processes are in place*. Table 4.1 mentions the lack of mitigation measurements. In the original model the situation *Data is encrypted* was included. However, this situation does not accurately represent multiple methods of data protection. Because of this the situation was adapted to show multiple means of data protection: *Data security and breach policies are in place.*

Indicators such as *Amount of closed issues* and *Commits* are relevant in a relative context, as shown by the results in Table 4.1 and 4.3. Because of this these have been changed to show a relative relation: *Relative amount of closed issues* and *Amount of commits relative to earlier amount of commits*. The time units

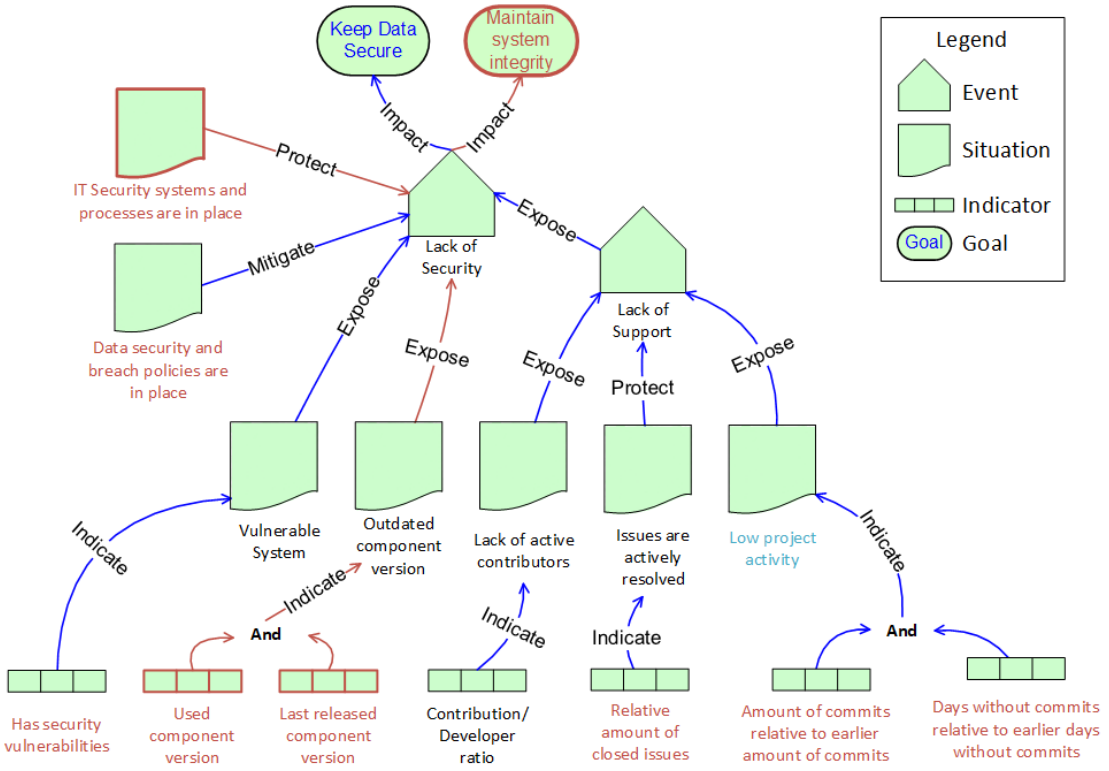


Figure 4.1: Final iteration of the Security Risks model. Changes are colored in orange. Clarifications are colored in aqua. Removed symbols and relations are not displayed.

have also been removed as those were degrading the clarity of the model and were too specific for a graphical representation of the risk.

For the situation *Outdated component version* two indicators were added: *Used component version* and *Last released component version*. As the *Outdated component version* is measurable by collected data these indicators add to the completeness of the model. This addition is based on the results shown in Table 4.4. The situation *Outdated component version* itself was changed to directly expose *Lack of Security* as based on the results in Table 4.2.

As based on the results in Table 4.3 the indicator *Amount of Security Vulnerabilities* was changes to *Has security vulnerabilities* as one security vulnerability is enough to put a system at risk.

A non-secure system can not only be targeted for data theft, but also for system

takeover or other forms of malicious intent. As shown in Table 4.4 example forms of malicious intent are: malicious code injection, hostile system takeover or acting as the system. The goal of a secure system is to protect it from malicious intent. Because of this a goal has been added to the model: *Maintain system integrity*, which shows another side of the importance of a secure system.

The final change in the model is the removal of *Framework Changes* situation. This would expose the *Lack of Support* but at the same time be protected by *Low project activity* (previously *Low activeness*). This situation was referred to as "unlikely" (Table 4.3), or "not relevant" in the model's context (Table 4.1).

Several interviewees mentioned the prediction of security vulnerabilities. Perl et al. (2015) did research on the prediction of security vulnerabilities based on commits and achieved accurate results. However, this study results are only applicable on projects that had at least one Common Vulnerabilities and Exposures (CVE) identified in the past, limiting the applicability on most components found in the case studies. As also mentioned in Table 4.2 predicting security vulnerabilities will be outside of the scope of this model and furthermore outside the scope of this research.

4.2.2 | Copyright Compliance model Iteration

Figure 4.2 shows the final iteration of the Copyright Compliance model. In this model the results of the interviews, as shown in Section 4.1.2, are incorporated. The changes compared to the original model in Figure 3.2 will be discussed based on the interview results.

Similarly to the results of the Security Risks model the Copyright Compliance model lacked business and IT processes that help reduce risks, in this case non-compliance. Results shown in Table 4.6, 4.7 and 4.8 mention automatic validation of licenses, company policies for licenses and periodic (external) OSS compliance checks. These are summarized in two new situations: *Business OSS Policies are in place* and *IT processes are in place to ensure compliance.* These situations reduce the likelihood of non-compliance, and thus have a protect relation to the *Non-compliance* event.

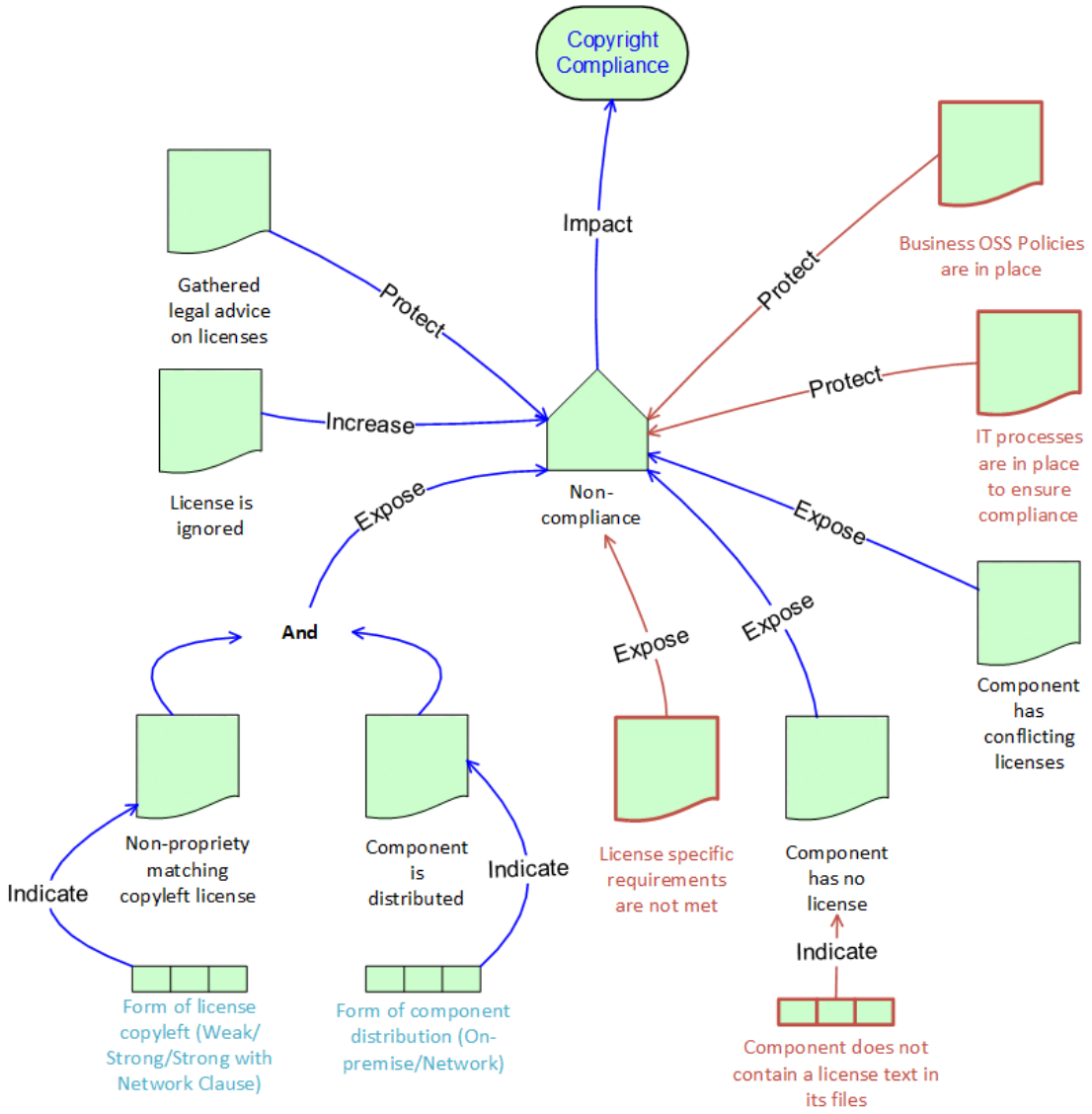


Figure 4.2: Iteration of the Copyright Compliance model. Changes are colored in orange. Clarifications are colored in aqua.

Table 4.6 mentions several requirements which might exist through the usage of different licenses. In the original model in Figure 3.2 copyleft, which requires the software to be open-sourced in a distributed form, is already included. Other requirements, such as license disclaimers, are not included. Because of this the *License specific requirements are not met* situation has been added. This covers non-copyleft requirements of a license which can cause non-compliance with the license.

The final change to the model is the addition of the *Component does not contain a license text in its files* indicator. As mentioned in the results in Table 4.9 situations that can be measured by automatic data collection should include an indicator.

As mentioned in Table 4.8 and 4.10 components can have multiple licenses. Having multiple license is either an extension of the model, as every situation can be looked at from the perspective of every license, or the license can be chosen from multiple licenses in which case only one license applies. In either case, the situation by itself does not expose *Non-compliance*, it exposes this event through the other situations. Because of this multiple licenses were not included in the model.

The deliberate removal of a license, as mentioned in Table 4.7, was also not included. The reason for this is that the removal of a license is similar to ignoring a license. Both count as willful infringement which is covered in the *License is ignored* situation. Because of this no separate situation was added.

4.3 | Decision Flow

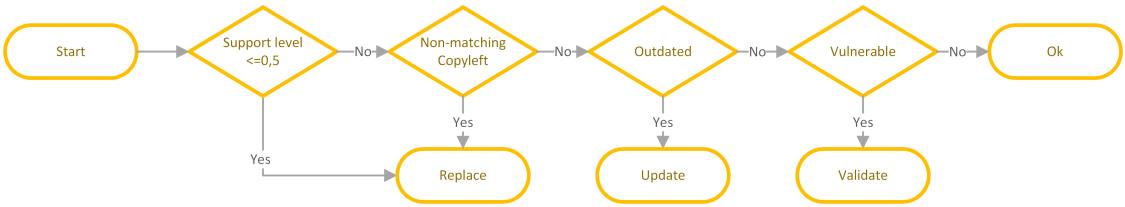


Figure 4.3: Decision tree for mitigation advice based on distributed applications.

Using the developed tool, different recommendations will be generated for each used component: Update, Replace, Validate or Ok. These recommendations are

based on the iterated risk models discussed in Section 4.2. In Figure 4.3 the decision points for these recommendations are shown. Components are recommended to be updated when they are outdated, replacement is advised when the component lacks support or has a non-propriety matching license, and validation is advised in case there are vulnerabilities found in the component.

In Figure 4.3 the decision *Support Level <= 0,5* represents the *Lack of Support* event in the Security Risks model. The cutoff point of 0,5 is based on the research by (Coelho et al., 2018) which states that this point indicates a project support level which is similar to an unmaintained project. The decision point *Non-matching Copyleft* is relative to the distribution of an application which contains propriety code. Non-matching means a form of copyleft which is more restricting than the used distribution form. This decision point represents the *Non-compliance* event in the Copyright Compliance. The *Outdated* and *Vulnerable* decision points represent the *Lack of Security* event in the Security Risks model.

There is no definitive advice given based on found vulnerabilities. Vulnerabilities can exist in situations in which they are not easily exploitable, can be easily mitigated by updating the component or are not relevant because the OSS component has no way to access sensitive areas of the system. Sensitive areas, such as data or relevant areas for the system's integrity, can be protected by limiting the access of an application in a system. This advice is based on the results of the second interview on security risks (Table 4.2).

Furthermore, while we describe the option to take on the role of project maintainer in Section 2.2.2 as a mitigation strategy, no such advice was found in the case studies. Because of this we expect that this is no viable option cost wise, and that mitigation through replacement or updating is a more efficient alternative. Because of this, this strategy is not included in the mitigation advises as seen in Figure 4.3.

Based on the created risk mitigation advises and the collected data we can create an estimate of the costs required to carry out these advises. We can then compare these costs with the expected impact costs of the risks as described in Section 2.2.3, as the mitigation costs should be lower than the impact costs to mitigate the risks from a financial risk management perspective.

4.4 | Case Studies

4.4.1 | Case A

Case A contains 18 applications. In these applications several identified risks contain Mitigation Effort (ME) estimations in case documentations. These estimations are based on risks identified by consultants that worked on this case. As this research managed to identify risks that are not identified in the original engagements, not all identified components with risks contain ME estimations to resolve these risks.

Based on the identified estimations we develop an approach for estimating fair value adjustments. This approach will be outlined in Section 4.4.1.1. The case findings are outlined and described in Section 4.4.1.2 and 4.4.1.3.

4.4.1.1 | Understanding Value

We found that update costs for components which are up-to-date to their major version, i.e. the current version is v1.0.0 and the newest version is v1.3.2, are not significant enough to measure. Components that are behind in their major version, i.e. the current version is v1.3.2 and the newest version is v3.0.0, can have significant costs for updates, although we found that this is often still not the case. Only one component was found with update costs for a major version update: the Spring Framework[1]. This framework was encountered with different versions, both behind in major version, as well as behind but up-to-date with its major version. The latter did not have a significant update cost. In total seven component version were found which were both outdated based on their major version, and were recommended to be updated by EY. Six of these were estimated to be simple updates without any significant costs.

Interviews were conducted with consultants who were involved in this case. Based on these interviews it was validated that components that are not behind in their major version number often do not have significant update costs. It might be possible to get a more accurate prediction for update costs by using more data

[1]https://github.com/spring-projects/spring-framework

points besides the change in version number. Differences in functionality or LoC between versions might act as data points for these cost estimations. As we do not have enough data available on how these data points would relate to costs, and no way to measure functionality changes, we decided to not include those in our value estimations. Instead we only looked at differences in major versions between the used component version and the latest component version. As with a major version difference of zero, the costs to update are insignificant.

The most important data points that are missing to create value adjustments are: how often a component is called in an application, and as what type the component can be classified. Some examples of types of components are: frameworks, logging components, testing frameworks and utility tools. These types could be further extended to get a better idea of how much a component is integrated in the application. This information would help in understanding the effort that would be required to replace or update a component. At the same time, this could help in understanding how much exposure is created by vulnerabilities in these components. As a component that has no access to sensitive data might be less of a security thread than a component that directly touches sensitive data.

Based on the information that is available, we can estimate lower and upper bounds for update costs of a component. The maximum found costs for updating a component was three person-months. These costs were associated with an update of a framework which was expected to affect a major part of the codebase. This component was part of application 2. The maximum costs for replacing a component was found to be 200 person-hours for a component in application 3. While the minimum cost of updating a component is not significant, the minimum cost for replacing a component was found to be 40 hours for application 1. For application 2 a component replacement was estimated at 60 hours, but this replacements has a lower Rebuild Factor (RF) due to the size of the application. As costs are looked at from the perspective of the refactor effort, the component with an expected effort of 60 hours will be used as a minimum.

Based on the mitigation effort we can estimate the expected affected LoC and use this as a maximum in our estimations:

$$RF = \frac{E(Ch)}{SS * TF}$$

Java is good for 65% of this application's code, while C++ and C combined is used in 35% of the application. CAST Highlight (CAST, 2020b) does not distinguish between C and C++, although based on the case documentation we know that this application only uses C++. We weight the TF according to the percentage of code per programming language. The TF for used languages is shown in Appendix C. Based on this data, application 2 has a weighted TF of 0,326. This gives a RV of 131,86 (person-months) for the application.

$$RV = 404{,}6 * 0{,}326 = 131{,}86$$

Based on the RV we can estimate the maximum RF for updating a component to be 2,28%.

$$RF = \frac{3}{131{,}86} = 0{,}0228$$

Based on the calculated RV and the percentage of code that needs refactoring (RF) the RV after adjustment can be calculated accordingly:

$$V_2 = 132{,}21 * (1 - 0{,}0228) = 128{,}86$$

This implies a Mitigation Effort (ME) of $132{,}21 - 129{,}21 = 3$, thus aligning with the value estimation based on effort:

$$V_1 = 132{,}21 - 3 = 128{,}86$$

As security risks, which is the reason to update this component, have an average impact cost of $3,92m the probability of one data breach has to be less than 0,64% over a five year period[2] to not mitigate the risk from a cost perspective. As these security risks are based on publicly known vulnerabilities, we estimate that the chance of a security breach is higher than this percentage. Therefore the V_3[3] value estimation is irrelevant for the mitigation of security risks.

The minimum effort to replace a component is 40 hours, which is based on the same application. Using the same calculations this gives a RF of 0,0415%.

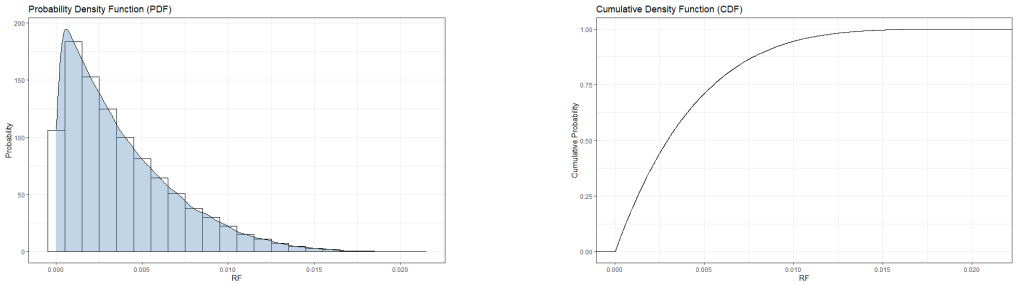
$$RF = \frac{0{,}055}{131{,}86} = 0{,}000415$$

[2]Assuming a $100,000 yearly developer salary.

[3]V_3 calculates risk according to expected impact over a five year period; see Section 2.2.2

The maximum cost to replace a component should at least have the same maximum as updating a component, as this means refactoring at least the same amount of LoC as when updating a component. As we did not find a maximum in replacement cost which is higher than the update costs, we assume the same RF maximum for replacement as for updating: 2,28%. We expect this cost to be lower than the expected impact cost of copyright infringement. This might depend on cost differences per country, but most importantly there should never be a recommendation to break the law. Hence we disregard V_3 for this risk.

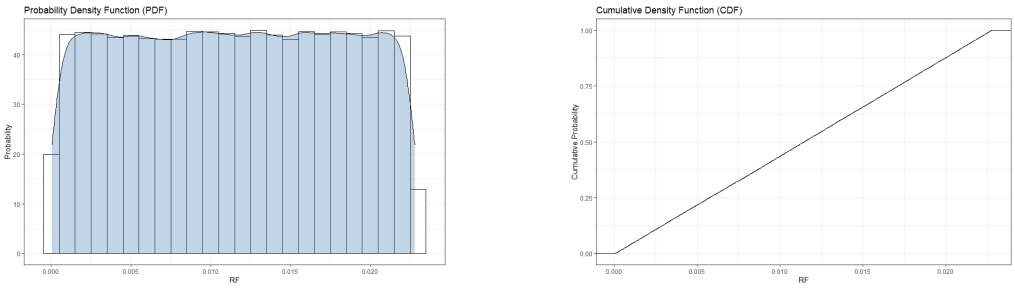
Based on the interviews it was confirmed that most components should have insignificant update costs. As it is expected that breaking changes will be caught by automated tests, and that the development time to work out any issues are not significant in the context of Mergers & Acquisitions (M&A). For replacing components, there is always some significant effort required. Based on the uncertainties of updating, but with multiple data points pointing towards a most likely scenario with zero costs, we can use the PERT distribution which can be seen Figure 4.4. For replacing components, we choose a uniform distribution as there is no available estimation on the most likely effort. This distribution can be seen in Figure 4.5. Using these values we can apply the Monte Carlo method to estimate the total RF of an application.



(a) Probability distribution of the RF for updating a single component.

(b) Cumulative probability of the RF for updating a single component.

Figure 4.4: Probability function for the updating of a single component. Generated using the Monte Carlo method.



(a) Probability distribution of the RF for replacing a component.

(b) Cumulative probability of the RF for replacing a component.

Figure 4.5: Probability function for the updating of a single component. Generated using the Monte Carlo method.

4.4.1.2 | General Findings

Table 4.11 shows the results of the used OSS components in Case A. In case documents costs for updates and component replacements are included. This data is used as the first step in understanding update costs for components.

It is expected that the amount of OSS components varies based on the primary language(s) used in an application. Some languages are more based around the usage of OSS components, such as JavaScript, while others less, such as C#, as can be seen in Table 4.11. What stands out in the case data is that the amounts of used OSS components varies widely between applications written in the same languages. For example application 11 and 12 are both written in Java with around one million LoC. The difference in OSS components however, is a relative difference of factor seven. It is possible that CAST Highlight (CAST, 2020b) is unable to properly identify OSS components in certain applications, or the purpose of the application might differ in such a way that it requires more or less OSS components.

In some applications CAST Highlight (CAST, 2020b) incorrectly identifies certain OSS components. An example of this is application 14 seen in Table 4.11. In this application libraries of the .NET framework were identified as OSS components. While the .NET framework is open source, most libraries found in this application are part of this framework and should not be identified as individual components. These are also not available as separate components on GitHub and thus cannot be identified with their own risks.

Table 4.11: Findings for Case A. Primary languages are the used languages that represent over 75% of the code of the application. Primary languages are ordered by their size.

	Application			**Identification**		**Advice**			**Findings**						**Findings (Identified)**		
Application	**KLoC**	**Primary Language(s)**	**Components**	**Identified Repositories**	**Non-identified Repositories**	**Update**	**Replace**	**Validate**	**Outdated Components**	**Outdated Major Versions**	**Vulnerable Components**	**Weak Copyleft Licenses**	**Strong Copyleft Licenses**	**Strong Network Copyleft Licenses**	**Unlicensed Components**	**Unsupported Components**	**Supported Components**
1	137,436	Java	243	22	221	69	10	3	75	23	7	5	5	0	2	5	17
2	404,583	Java/C/C++	256	14	242	188	12	6	198	78	38	5	12	0	0	0	14
3	1345,366	C#	59	21	38	34	5	1	37	15	11	2	1	0	0	4	17
4	349,442	C#	1358	891	467	751	64	11	776	422	40	0	0	0	1	65	826
5	3,32	JavaScript	904	701	203	589	46	7	619	418	38	0	0	0	1	46	655
6	6,311	JavaScript	850	658	192	556	46	5	589	401	37	1	0	0	1	46	612
7	6,034	JavaScript	1440	1071	369	878	77	9	931	527	53	0	0	0	3	77	994
8	7,443	JavaScript	832	646	186	538	44	6	567	383	43	0	0	0	1	44	602
9	1,408	JavaScript	1177	900	277	731	56	6	765	485	50	1	0	0	1	56	844
10	108,16	JavaScript/PHP	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	972,209	Java	225	44	181	185	15	1	195	50	32	2	7	0	0	8	36
12	1158,496	Java	1928	1250	678	1117	95	18	1156	579	50	1	0	0	1	97	1153
13	396,972	Java	1022	688	334	639	53	5	662	339	34	0	1	0	0	53	635
14	11,151	C#	45	5	40	41	0	0	41	30	4	2	0	0	0	0	5
15	9,058	Java/Typescript	1335	936	399	935	69	6	979	546	53	3	1	0	1	70	866
16	229,999	C#	133	18	115	92	2	2	93	57	12	3	1	0	0	1	17
17	360,322	Java/JavaScript	73	32	41	57	4	2	60	24	13	1	0	0	0	4	28
18	5,187	Scala	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

4.4.1.3 | Value Determination

Table 4.12: Relevant application data points for mitigation strategies in Case A.

Application	Major Updates	Replacements	Weighted TF	RV	Minimum ME	ME (1st quartile)	Median ME	Mean ME	ME (3rd quartile)	Maximum ME	Variance	Standard Deviation
1	19	10	0,35	48,31	4,25	8,17	8,99	9,00	9,82	14,26	10,30	3,21
2	71	12	0,33	131,85	35,18	50,49	53,60	53,66	56,79	75,43	166,73	12,91
3	13	5	0,33	439,80	17,01	41,14	46,63	46,76	52,24	82,87	448,85	21,19
4	408	64	0,32	113,28	211,12	252,04	258,37	258,43	264,76	301,40	832,53	28,85
5	400	46	0,31	1,02	1,76	2,03	2,08	2,08	2,14	2,44	0,05	0,22
6	378	46	0,31	1,94	3,21	3,70	3,80	3,80	3,90	4,41	0,15	0,38
7	494	77	0,31	1,85	4,44	4,99	5,10	5,10	5,22	5,85	0,20	0,45
8	366	44	0,31	2,28	3,61	4,21	4,32	4,32	4,44	5,05	0,21	0,46
9	460	56	0,31	0,43	0,87	1,01	1,03	1,03	1,06	1,19	0,01	0,10
10	0	0	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
11	185	15	0,35	337,36	224,57	283,42	294,86	295,04	306,49	371,03	2200,73	46,91
12	550	95	0,35	408,83	1130,19	1270,67	1297,68	1297,89	1325,04	1469,80	11830,15	108,77
13	320	53	0,34	135,45	206,66	239,77	246,62	246,65	253,44	292,71	761,60	27,60
14	30	0	0,33	3,67	0,18	0,37	0,42	0,42	0,46	0,72	0,03	0,17
15	529	69	0,34	3,12	7,54	8,53	8,72	8,72	8,92	10,06	0,65	0,81
16	51	2	0,32	74,73	8,75	14,91	16,15	16,19	17,41	24,62	25,87	5,09
17	22	4	0,34	121,50	6,56	14,04	15,65	15,69	17,32	26,41	40,65	6,38
18	0	0	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA

As can be seen in Table 4.12 most of the applications with OSS components have a higher ME in their best case mitigation scenario than their RV. What this means is that it is estimated, in the best case, to be more costly to mitigate all the issues than to rebuild the application in the same way. Since most of these applications require to solve issues of multiple hundreds of components, from which more than 10% require replacements, which have a higher mean effort than updates, this is not unrealistic. It is important to note here that RV does not equal the fair value, meaning that the value adjustment, ME, can be lower than the fair value of the application.

4.4.2 | Case B

This case only has three applications that contain a large amount of OSS components. These applications are primarily written in JavaScript/Typescript. This case also contains some applications that are written in languages that are not encountered in other cases such as Swift and Kotlin.

4.4.2.1 | General Findings

As can be seen in Table 4.13, of the 18 applications in Case B 13 contained identified OSS components. Most of these applications contained less than 100 components, which compared to Case A can be identified as an overall low usage of OSS components. As the overall component usage is relatively low, it is a question on how well CAST Highlight (CAST, 2020b) is able to detect components. Some applications are written in lesser used languages which might cause less OSS component to exists for these languages. Examples of this are application 7 and 9, which are written in Swift and Kotlin respectively. These applications are also smaller based on their LoC than most other applications, which might be related to a smaller usage of OSS components. However, other applications such as applications 3, 12 and 13 are written in languages that were using a higher relative amount of components in Case A. This might be explained by their lower amount of LoC, which creates a different need for OSS components, or because they were written by a different company with different OSS policies.

Table 4.13: Findings for Case B. Primary languages are the used languages that represent over 75% of the code of the application. Primary languages are ordered by their size.

	Application			Identification		Advice			Findings						Findings (Identified)		
Application	**KLoC**	**Primary Language(s)**	**Components**	**Identified Repositories**	**Non-identified Repositories**	**Update**	**Replace**	**Validate**	**Outdated Components**	**Outdated Major Versions**	**Vulnerable Components**	**Weak Copyleft Licenses**	**Strong Copyleft Licenses**	**Strong Network Copyleft Licenses**	**Unlicensed Components**	**Unsupported Components**	**Supported Components**
1	148,471	PHP	22	9	13	15	2	0	16	5	9	1	0	0	0	2	7
2	24,301	C#	1	1	0	1	0	0	1	1	0	0	0	0	0	0	1
3	59,888	Java	3	2	1	1	1	0	2	0	0	0	1	0	0	0	2
4	863,028	JavaScript	1317	904	413	954	79	8	1019	575	68	8	8	1	14	72	832
5	18,334	PHP	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0,477	Python	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	1,971	Swift	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	5,459	Python	4	2	2	2	0	0	2	1	0	0	0	0	0	0	2
9	1,403	Kotlin	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	514,175	PHP/JavaScript	79	27	52	46	5	0	49	22	4	0	2	1	5	2	25
11	258,763	JavaScript/PHP	4	2	2	1	3	0	3	3	1	0	1	1	2	1	1
12	78,289	C#	2	1	1	2	0	0	2	1	0	0	0	0	0	0	1
13	87,268	JavaScript	20	11	9	10	4	0	14	7	5	1	0	0	0	4	7
14	1382,299	JavaScript/PHP	2107	1426	681	1431	127	14	1522	886	98	15	10	3	8	118	1308
15	108,114	PHP	5	4	1	5	0	0	5	3	0	0	0	0	0	0	4
16	8,105	Typescript	1033	838	195	545	67	10	573	346	35	0	0	0	0	69	769
17	15,57	JavaScript/PHP	13	4	9	10	0	1	10	2	1	0	0	0	0	0	4
18	108,032	C/C++	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Not all components use some form of releasing versions through Git or GitHub, such that these are available on GitHub through the Git Tag system or the GitHub 'Release' system. Some components use these tools but these are not as up-to-date as the releases reported on their distribution platform. When either of these situations is the case, such that no up-to date release version information is available on GitHub, components cannot properly be identified on GitHub. This is the case because the tool matches the latest known release reported by the CAST Highlight (CAST, 2020b) output to the information on GitHub. This can cause issues in identifying repositories as the repository might be found, but the release information cannot be matched as this data is outdated on GitHub. Meaning that the repository will not be correctly identified, and thus will be reported as a non-identified repository. This causes a lower GitHub identification ratio throughout all applications than what we expect with up-to date information.

4.4.2.2 | Value Determination

As can be seen in Table 4.14 it is costly to not be a step ahead of issues with OSS components. For example application 4 requires almost half of its components to be updated to a new major version and is recommended to replace 79 of its components. As can be seen by the 25-75% range of ME this will likely cost around three times the cost of building the application. Even though these estimates are based on rough estimates, the costs of updating alone is expected to surpass 542 person-months in 25% of the cases, which is based on a more certain estimate than replacements and is still two times RV.

Table 4.14: Relevant application data points for mitigation strategies in Case B.

Application	Major Updates	Replacements	Weighted TF	RV	Minimum ME	ME (1st quartile)	Median ME	Mean ME	ME (3rd quartile)	Maximum ME	Variance	Standard Deviation
1	5	2	0,35	51,43	0,34	1,73	2,14	2,15	2,56	4,62	1,93	1,39
2	1	0	0,33	8,00	0,00	0,01	0,02	0,03	0,04	0,17	0,00	0,06
3	0	1	0,35	21,14	0,00	0,12	0,24	0,24	0,36	0,48	0,03	0,17
4	545	79	0,31	268,75	689,13	781,91	799,04	799,14	816,25	916,43	5288,25	72,72
5	0	0	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
6	0	0	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
7	0	0	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
8	1	0	0,37	2,02	0,00	0,00	0,01	0,01	0,01	0,04	0,00	0,02
9	0	0	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
10	17	5	0,33	171,73	7,92	18,55	20,83	20,88	23,17	36,29	83,05	9,11
11	1	3	0,33	85,47	0,15	2,54	3,26	3,25	3,96	6,67	4,47	2,11
12	1	0	0,33	25,77	0,00	0,03	0,08	0,10	0,14	0,55	0,04	0,20
13	4	4	0,32	27,62	0,27	1,40	1,68	1,68	1,96	3,30	0,95	0,98
14	837	127	0,33	455,40	1891,10	2072,54	2108,93	2109,02	2145,23	2360,02	22591,14	150,30
15	3	0	0,35	37,56	0,01	0,27	0,40	0,43	0,56	1,56	0,29	0,53
16	327	67	0,36	2,90	4,87	5,67	5,82	5,82	5,97	6,88	0,41	0,64
17	2	0	0,33	5,08	0,00	0,02	0,03	0,04	0,05	0,17	0,00	0,06
18	0	0	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA

4.4.3 | Case C

This case consists of the most applications. There are all analyzed on their OSS component risks. Value estimations are made based on the approach described in Section 4.4.1.1 are are outlined in Section 4.4.3.2.

4.4.3.1 | General Findings

Notably in the results of Case C is that the languages C, Objective C and C++ are not distinguishable by CAST Highlight (CAST, 2020b). In Table 4.15 it can be seen that applications using these languages have a combination of these languages as their *Primary Language(s)*. This is most likely due to the same syntax of these languages. While this does not cause the identification of OSS components to change, it does affect the effort per LoC for the application, as these value differ between the overlapping languages. This is further discussed in Section 4.4.3.2.

Table 4.15: Findings for Case C. Primary languages are the used languages that represent over 75% of the code of the application. Primary languages are ordered by their size.

	Application			Identification		Advice			Findings						Findings (Identified)		
Application	KLoC	Primary Language(s)	Components	Identified Repositories	Non-identified Repositories	Update	Replace	Validate	Outdated Components	Outdated Major Versions	Vulnerable Components	Weak Copyleft Licenses	Strong Copyleft Licenses	Strong Network Copyleft Licenses	Unlicensed Components	Unsupported Components	Supported Components
1	14,219	C/C++	4	0	4	2	0	1	2	0	4	0	0	0	0	0	0
2	30,334	C/C++	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	14,062	C/C++	4	2	2	2	0	1	2	1	1	2	0	0	0	0	2
4	57,269	Ruby/JavaScript	10	2	8	4	2	0	6	4	0	0	2	0	0	0	2
5	0,263	Typescript/JavaScript	1390	923	467	758	47	13	776	429	29	0	0	0	1	58	865
6	10,899	C/C++	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	23,97	Java	5	0	5	4	0	0	4	1	0	0	0	0	0	0	0
8	455,814	C/C++	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	7777,341	C/C++	1032	739	293	690	52	6	718	413	44	3	6	0	2	59	680
10	97,889	C#	104	11	93	71	2	2	72	17	7	1	2	0	0	0	11
11	4,35	Python	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	876,055	C#	1388	862	526	871	50	10	893	442	50	2	1	1	2	63	799
13	27,957	Typescript	1275	969	306	844	54	11	871	516	52	0	0	0	0	72	897
14	155,481	C#	85	18	67	69	1	0	70	37	2	0	0	0	0	1	17
15	94,956	C/C++	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0
16	2201,544	C/C++/C#	392	40	352	290	8	3	296	123	15	3	4	4	0	0	40
17	24,502	C#/JavaScript	1036	725	311	741	36	4	759	498	32	0	0	0	0	48	677
18	75,013	C/C++	5	0	5	3	0	1	3	0	1	0	0	0	0	0	0
19	415,947	C#	2322	1390	932	1475	77	770	1517	892	70	0	2	2	0	99	1291
20	90,224	JavaScript/C#	30	16	14	23	1	0	23	4	1	0	0	0	0	3	13
21	55,513	Objective C/C++/C/C++/C#	13	6	7	6	3	0	8	5	0	0	0	0	0	3	3
22	535,116	C/C++	5	1	4	2	1	0	3	1	0	1	1	0	0	0	1
23	3641,832	C/C++	11	2	9	9	0	0	9	5	0	0	0	0	0	0	2
24	25,656	C#/C/C++	257	204	53	193	13	51	198	116	18	0	0	0	0	14	190
25	1371,947	C/C++	30	7	23	14	2	1	15	6	2	0	1	1	0	0	7
26	28,86	Ruby	207	159	48	147	10	2	153	44	25	0	4	0	0	10	149
27	179,112	Ruby/JavaScript	258	178	80	185	15	4	195	63	37	0	5	0	1	15	163
28	102,608	C/C++	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0
29	55,336	C#	19	11	8	14	1	0	14	6	3	0	0	0	0	1	10
30	30,717	C/C++	7	1	6	5	0	2	5	1	2	2	0	0	0	0	1

4.4.3.2 | Value Determination

Table 4.16: Relevant application data points for mitigation strategies in Case C.

Application	Major Updates	Replacements	Weighted TF	RV	Minimum ME	ME (1st quartile)	Median ME	Mean ME	ME (3rd quartile)	Maximum ME	Variance	Standard Deviation
1	0	0	0,32	4,51	0	0	0	0	0	0	0	0
2	0	0	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
3	0	0	0,32	4,44	0	0	0	0	0	0	0	0
4	3	2	0,33	18,92	0,03	0,50	0,64	0,65	0,79	1,45	0,21	0,46
5	411	47	0,337	0,088	0,16	0,18	0,19	0,19	0,19	0,22	0,00	0,02
6	0	0	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
7	1	0	0,35	8,46	0,00	0,01	0,02	0,03	0,05	0,18	0,00	0,07
8	0	0	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
9	390	52	0,32	2474,75	4333,78	5005,73	5137,14	5137,64	5268,04	6019,77	291554,65	539,96
10	13	2	0,33	32,22	0,68	2,00	2,32	2,33	2,64	4,58	1,58	1,26
11	0	0	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
12	418	50	0,33	288,13	522,19	606,60	622,14	622,23	637,93	721,67	4077,33	63,85
13	497	54	0,35	9,86	20,65	24,13	24,71	24,71	25,28	28,31	6,02	2,45
14	37	1	0,33	51,71	3,71	7,12	7,83	7,86	8,56	12,74	8,39	2,90
15	0	0	0,32	30,71	0	0	0	0	0	0	0	0
16	117	8	0,32	714,18	266,76	363,43	382,41	382,72	401,50	512,00	6171,96	78,56
17	483	36	0,33	8,16	15,54	17,88	18,33	18,33	18,77	21,18	3,26	1,81
18	0	0	0,32	23,77	0	0	0	0	0	0	0	0
19	859	77	0,34	139,34	518,26	567,00	577,41	577,46	587,85	641,46	1562,73	39,53
20	4	1	0,31	28,29	0,05	0,56	0,75	0,75	0,93	1,91	0,37	0,61
21	3	3	0,35	19,30	0,12	0,71	0,88	0,88	1,05	1,87	0,32	0,57
22	0	1	0,32	169,58	0,01	0,97	1,94	1,94	2,90	3,87	1,86	1,36
23	4	0	0,32	1154,10	0,41	12,04	16,75	17,51	22,09	58,45	387,57	19,69
24	111	13	0,33	8,53	3,55	4,63	4,86	4,87	5,10	6,41	0,84	0,92
25	6	2	0,32	436,14	3,26	16,13	19,76	19,87	23,48	43,43	170,10	13,04
26	42	10	0,34	9,69	1,49	2,46	2,65	2,65	2,85	3,89	0,59	0,77
27	60	15	0,32	58,09	14,85	21,78	23,16	23,19	24,59	32,27	31,17	5,58
28	0	0	0,32	32,67	0	0	0	0	0	0	0	0
29	6	1	0,33	18,39	0,09	0,49	0,62	0,63	0,75	1,48	0,21	0,46
30	1	0	0,32	9,76	0,00	0,01	0,03	0,04	0,05	0,21	0,01	0,08

Table 4.16 shows the estimated ME for applications in Case C. In applications that contain a combination of C and C++ or Objective C and C++, both were weighted as 50% relative to the total usage of these two languages for the calculation of TF. As neither the case documentation, nor the CAST Highlight (CAST, 2020b) output distinguish between C and C++, and Objective C and C++, it is unknown what the exact usages are of C, C++ and Objective C individually.

As can be seen in Table 4.16, in application 5, 9, 12, 13, 17 and 19, the minimum ME exceeds the RV of the application. As the fair value of the application can be higher than the RV, this does not mean that the costs exceed the value of the application. It does show that resolving all issues in these applications is more costly than rebuilding the application in the current state.

5

Discussion

In this chapter the threats to validity and limitations of this study will be discussed. Validity will be discussed from an internal and external perspective.

5.1 | Threats to Validity

5.1.1 | Internal Validity

As became clear from the interview results, *Vulnerable System* (Figure 4.1) could be more adequately measured. Firstly, as explored by Pashchenko, Plate, Ponta, Sabetta, and Massacci (2018) not all dependencies are deployed. This means that certain vulnerabilities are not exposed and might not be relevant in measuring whether a system is vulnerable or not. Secondly, not every vulnerability is exploitable in the context of the application and some vulnerabilities require a certain amount of access to the system to exploit. These criteria could give a more complete indication of whether a system is vulnerable or not. This research looks for vulnerabilities in multiple vulnerability databases. While this might not give a complete view of exploitability, it does give a good indication on whether or not an application is potentially at risk to data loss or loss of system integrity.

The amount of OSS components in some applications deviated marginally from other applications. Some applications did not have any OSS components, while these still contained thousands LoC. Other applications with small codebases

contained many OSS components. As this study relies on the output of CAST Highlight (CAST, 2020b) there is no way to validate the completeness or accuracy of the reported OSS components in an application. While this might limit the completeness in the reported risks through the developed tool, it does not limit the researched approach to determining value in software through OSS component risk assessment.

5.1.1.1 | Repository Identification Ratio

Table 5.1: Identification ratio of GitHub repositories from OSS components.

Case	Identification Ratio
Case A	66.5%
Case B	67.8%
Case C	63,4%
Average	65,9%

The tool is able to identify the repositories of around 66% of the components, as can be seen in Table 5.1. We expect this to be slightly higher in reality as some components are part of a larger framework, so they do not exist as their own component, but are identified as such. Because of this, applications which use large frameworks can have a relative low identification ratio compared to other applications. Non-identified components can cause risks to not be identified. As data points are missing for these components which are required to create a complete understanding of the risks. While we do not expect this to change the outcome of this study it is an issue to explore in future works.

5.1.1.2 | Accuracy of Mitigation Effort

Mitigation Effort (ME) is based on a limited data set of Case A. While this uncertainty is included in the estimations it is possible that maximum costs are not accurate, and that these are either higher or lower. If this is the case, the value adjustments will change accordingly. As there is no ground truth to validate these findings, there is no statistical evaluation possible of these estimations. However,

we provide an approach which is consistent and is improvable by providing more accurate data. Future works can validate the results of this method against a ground truth and improve this method with different data points.

5.1.2 | External Validity

This research is based on data availability through CAST Highlight (CAST, 2020b), limiting the generalizability of the created approach in case this data is not available. The approach to identify risks and create fair value adjustments is, however, generalizable, as long as it is understood which OSS components are used in an application.

5.2 | Limitations

This researched created the initial steps in understanding the impact of OSS component risks on the value determination of software. In this chapter the limitations of this research will be discussed.

5.2.1 | Risk of Failure

In Section 2.1 the risk of failure was discussed. This risk was not further incorporated in the risk models discussed in this research. The reason for this is that the cost for risk of failure is dependent on the application and the functionality of the component. This makes it less generalizable than the costs for security risks, lack of support or copyright infringement. Due to this the risk of failure was deemed to be outside of the scope of this research. For this reason the understanding of this risk in relation to value determination is still limited.

5.2.2 | Tool Limitations

To assess the risks of OSS components in applications a tool was created based on the created qualitative models in this research. The limitations of this implementation are discussed in this section.

5.2.2.1 | Accessibility of Data

Git repositories can be hosted on a multitude of platforms. Of these platforms, GitHub is the largest host of repositories (Kalliamvakou et al., 2014). This study primarily aimed to identify risks based on data of GitHub. Therefore risk findings of repositories not hosted on GitHub are limited.

The means of identifying repositories in the model implementation is also limited. A subset of components reported by CAST Highlight (CAST, 2020b) are hosted on GitHub but could not be retrieved. As CAST Highlight (CAST, 2020b) only reports a limited amount of information about the repository this information is not always enough to identify the correct repository. This again limits the completeness of the risk assessment in software.

Some application do not contain any OSS component according to CAST Highlight (CAST, 2020b). While this might be expected for applications that contain only a few thousand LoC, it is a rare occurrence for applications with thousands LoC. Based on the results of the case studies it is expected that there are missing OSS components in the output of CAST Highlight (CAST, 2020b) which are therefore not included in the risk assessment. This limits the completeness of risk assessment of the application as a whole.

5.2.2.2 | Identification of Situations

In the RiskML models shown in Section 4.2 some situations lack indicators. While information on some situations can be gathered through non-automated means, such as through interviews with developers, this is not the case for all of them. Situations such as *Component has conflicting licenses* are important to determine whether an application is compliant with certain licenses. The issue with these situations is that the implementation of these models is not capable to identify whether these situations are fulfilled. This means that it cannot be determined whether a component is at risk based on these situations. While these situations help in the understanding of the risks, and therefore the risk assessment, the tool is limited in assessing the risks completely due to these missing data points.

5.2.2.3 | Generalization Capability of Machine Learning

As discussed in Section 3.3 to determine the support level of a component the Random Forest implementation of Coelho et al. (2018) was used. Any form of machine learning can suffer from bias due to overfitting or underfitting the model. To reduce this form of bias a form of cross-fitting was applied on the dataset: K-Fold Cross Validation. In this form of cross-fitting the model is trained with a percentage of the complete training set, while the other set is used for validation. This confirmed the accuracy of the model. Furthermore the differently trained models were applied to new data from the case studies. An aggregation of the results (supported or non-supported) was reported as the support level for the OSS component. In this way the certainty of the prediction on the available data could be estimated. A high value indicates a high likelihood that the component is actively maintained, while a low value means a high likelihood that the component is no longer maintained. As the accuracy of a machine learning tool is never 100%, there is a limitation in the outcomes of this model due to the possible inaccuracy of the machine learning model.

6

Conclusions

This research aimed to provide an answer to the question: *How can Open Source Software (OSS) Component Risk Assessment be used for the Value Determination of Software?* In this section we will provide an answer to this question through the formulated sub-questions.

What risks are involved in using OSS components? The following risks exist in using OSS components: security risks, risk of failure, loss of support and copyright infringement.

How can OSS component risks impact fair value? Risks in using OSS components can impact the fair value by possible damages to a company. Alternatively fair value adjustments can be based on the mitigation costs of risks. Possible damages of risks can happen through security breaches in case of security risks, lawsuits and possible fines or injunctive relief in case of copyright infringement, and maintenance costs in case of loss of support.

How can OSS component risks be identified? Identification of a risk can be done by looking at different situations which can exist within a company or an application, combined affecting the likelihood and/or severity of risk occurrences. We found seven situations that can help in assessing security risks and eight situations to assess copyright compliance risks.

How can the costs of mitigating OSS component risks be quantified? Costs of mitigating risks in components can be quantified by looking at the amount of a system which has to be rebuild for each OSS component risk. These values can be used to calculate a best case, most likely and worst case total amount of effort

that is required to mitigate those risks, which effort equals the mitigation cost.

How can Open Source Software (OSS) Component Risk Assessment be used for the Value Determination of Software? Risk assessment can be done by identifying risk situations in OSS components. The situations that identify risk events, in combination with an understanding of how business processes help mitigate and protect against damages from these risks, help in understanding which components and how these components bring risk to an application. When at risk components are identified, the value of software can be adjusted based on the potential damages or mitigation costs for these risks. Components that are at risk based on a lack of support or copyright compliance issues, should be replaced. While lack of support can be mitigated by taking up the maintainer role, this is estimated to not be feasible in terms of costs. As when the expected impact cost is higher than the mitigation cost, it is better to mitigate the risk. Security vulnerabilities can sometimes be resolved by upgrading a component. If the latest version still contains security vulnerabilities migrating to a similar component without vulnerabilities might be a solution.

We propose a method to quantify value adjustments based on a best case, most likely and worst case effort estimation. By applying the Monte Carlo method on these values, with a distribution matching the certainty of this data, an effort range can be generated. By deducting these effort values from the fair value of software, a new value range can be determined for software.

6.1 | Achieved Aims and Objectives

The aim of this research was to fill the gap between OSS component risks and how this impacts the value determination of software. We filled this gap by creating two qualitative models that display the relations between different risk indicators and the goals of an application in relation to these risks. We created a tool to further assess the risks in case studies based on the qualitative models. This tool was created using R: A Language and Environment for Statistical Computing (R Core Team, 2020). With this tool we were able to collect data on the indicators for the researched risks. These indicators were used to assess whether a component

was at risk or not. We recommended two courses of action for components at risk: update or replace. A separate recommendation was given for vulnerabilities: validate. Components that could resolve their issues by updating were recommended to do so, as updating is cheaper than replacing. Replacement was advised in case a component had a support score lower or equal to 0.5 which meant that the component was no longer maintained. In case there were copyleft licenses found, and the application was distributed, the tool recommends to replace the component.

6.2 | Contributions

The overall contribution of this thesis is a designed and created approach to determine fair value adjustments in the context of software acquisitions based on software and data analytics. In the course of doing so an improved understanding of the relationship between OSS components and software value was developed. This understanding contributes to the literature of value estimation and delivery in SDD, agile methods, continuous delivery and continuous integration.

Two conceptual models were created to increase the understanding of security and copyright compliance risks in OSS components. These models further a quantifiable approach in determining whether components are at risk. We created a consistent method to determining value impacts of risks based on their mitigation strategies.

To gather and analyze OSS risks data in the case studies of this research a tool was created. This tool is able to gather and analyze data from different sources and combines these in a recommendation on risk mitigation per used component. These recommendation are translated into fair value adjustments based on their mitigation effort. This tool is a contribution towards EY as the tool is transferred in full to EY whom also included its use in their business processes.

6.3 | Future Work

As described in Section 5.2, this study comes with several limitations based on the availability of data. This study looked at how risk assessment of OSS components

can be used for value determination of software, in future work this approach can be further extended to gather further quantifiable results.

6.3.1 | Repository Identification

This research made the first steps in filling the gap between OSS component risks and the impact of these risks on value determination of software. As described in Section 5.2.2.1, not every GitHub repository can be identified. At the same time identification of releases can be troublesome as addressed in Section 4.4.2. A proposed solution for future works is identification through distribution platforms such as NPM or Maven. CAST Highlight (CAST, 2020b) contains the URLs for the identified components on the respective distribution platform. Currently this information is not available through their API, but it is available in their online platform. These distribution platforms often contain the URL to the correct GitHub page, including ones that currently cannot be identified. Using this information can also help in identifying proper release information, as release information from both the distribution platform and GitHub can be collected and validated against each other. As release information is not always up-to-date on one of these platform, using both will ensure the latest information.

6.3.2 | Fair Value Adjustments

This research provides a method of identifying risks in OSS components and creating an preliminary understanding of how this translates into value adjustments. However, it lacks understanding in how a component is used, and what the functionality of a component is. To create a more accurate understanding of value adjustments based on the identified risks a more detailed understanding of these topics is required. Understanding of usage can be done in the form of the amount of calls to the component in an application, or with a more simplistic approach by having a baseline of different types of components. Neither of these two things are currently known through CAST Highlight (CAST, 2020b) but might be available in the future.

For future work we propose the inclusion of the data points: type and usage of components. These data points can be used to determine a more accurate estimation of mitigation costs through updating or replacements of components. The method which we propose in Section 4.4.1.1 can be used with more accurate estimations per component to create a consistent and accurate range of mitigation effort for OSS component risks in applications. Identification of the type of component can be done by classification based on the contents of a README file. Prana, Treude, Thung, Atapattu, and Lo (2019) did work on such a GitHub classification method which can be further researched in the context of value determination of software. Usage of a component can be done by scanning the code of an application. By scanning the code for patterns that relate to used components it might be possible to get a size estimation of the component in relation to the application.

6.3.3 | Component Usage Identification

In the current state CAST Highlight (CAST, 2020b) does not include a way to retrieve component usage information. CAST Application Intelligence Platform (CAST, 2020a) has the option to scan an application for occurrences of keywords. A possible way to identify how often a component is used is by extracting the function names from a component's source code. These can then be included as keywords in an application (re-)scan. This will give an initial insight in the usage of components in an application in an automated way.

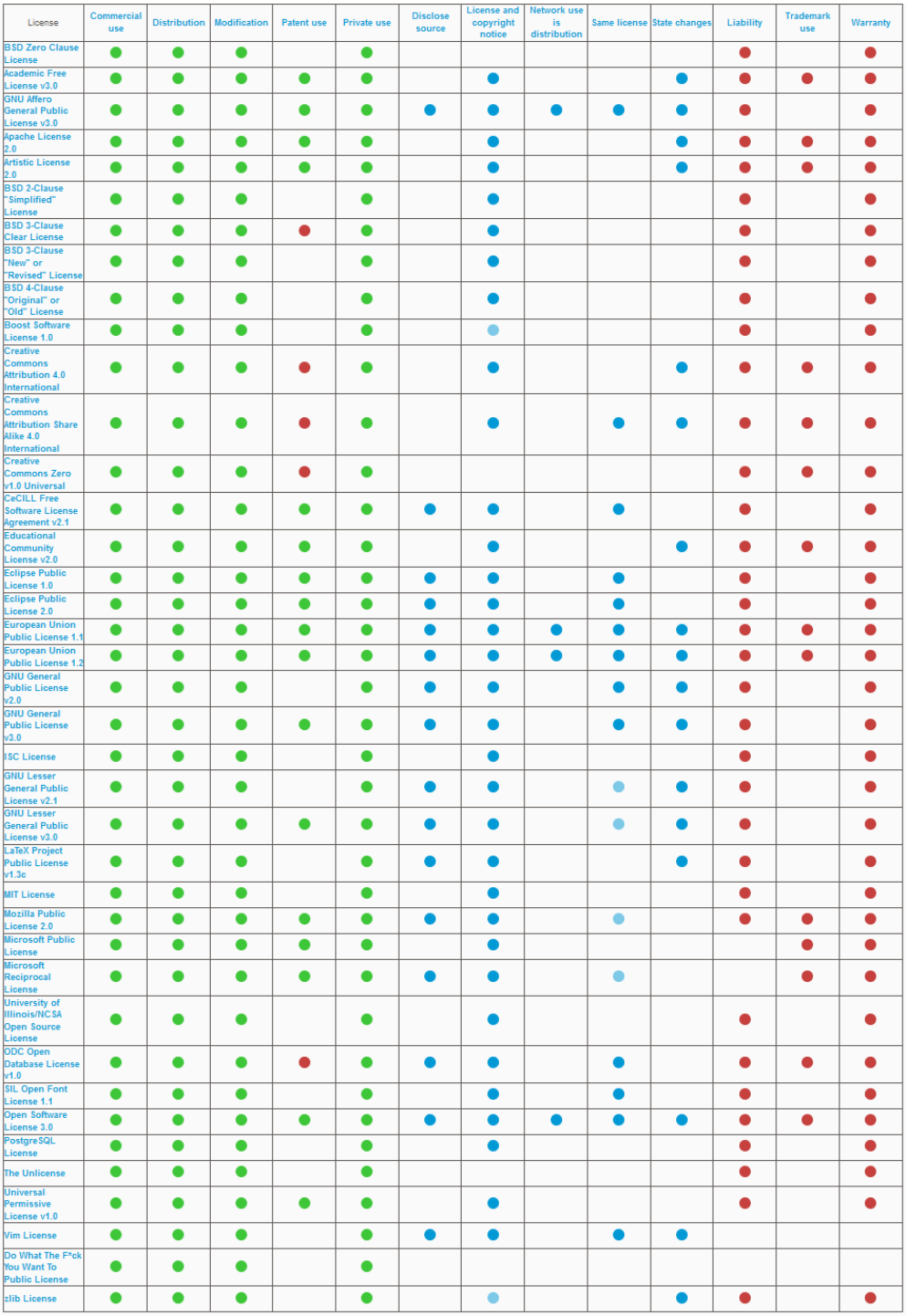
6.4 | Final Remarks

As part of this research we developed a tool to assess the risks of used OSS components in an application. This tool was developed based on the CAST Highlight (CAST, 2020b) output which is a powerful, but limited, source of information on OSS risks. The developed tool is able to improve on this output by assessing risks, and recommending mitigation options for these risks. It has already been successfully applied in a running case during this research, and will be applied in more

cases in the future. Due to this success, the tool is being integrated in the standard business process of EY.

A

OSS Licenses



License	Commercial use	Distribution	Modification	Patent use	Private use	Disclose source	License and copyright notice	Network use is distribution	Same license	State changes	Liability	Trademark use	Warranty
BSD Zero Clause License	●	●	●		●						●		●
Academic Free License v3.0	●	●	●	●	●		●			●	●	●	●
GNU Affero General Public License v3.0	●	●	●	●	●	●	●	●	●	●	●		●
Apache License 2.0	●	●	●	●	●		●			●	●	●	●
Artistic License 2.0	●	●	●	●	●		●			●	●	●	●
BSD 2-Clause "Simplified" License	●	●	●		●		●				●		●
BSD 3-Clause Clear License	●	●	●	●	●		●				●		●
BSD 3-Clause "New" or "Revised" License	●	●	●		●		●				●		●
BSD 4-Clause "Original" or "Old" License	●	●	●		●		●				●		●
Boost Software License 1.0	●	●	●		●		●				●		●
Creative Commons Attribution 4.0 International	●	●	●	●	●		●			●	●	●	●
Creative Commons Attribution Share Alike 4.0 International	●	●	●	●	●		●		●	●	●	●	●
Creative Commons Zero v1.0 Universal	●	●	●	●	●						●	●	●
CeCILL Free Software License Agreement v2.1	●	●	●	●	●	●	●		●		●		●
Educational Community License v2.0	●	●	●	●	●		●			●	●	●	●
Eclipse Public License 1.0	●	●	●	●	●	●	●		●		●		●
Eclipse Public License 2.0	●	●	●	●	●	●	●		●		●		●
European Union Public License 1.1	●	●	●	●	●	●	●	●	●	●	●	●	●
European Union Public License 1.2	●	●	●	●	●	●	●	●	●	●	●	●	●
GNU General Public License v2.0	●	●	●		●	●	●		●	●	●		●
GNU General Public License v3.0	●	●	●	●	●	●	●		●	●	●		●
ISC License	●	●	●		●		●				●		●
GNU Lesser General Public License v2.1	●	●	●		●	●	●		◐	●	●		●
GNU Lesser General Public License v3.0	●	●	●	●	●	●	●		◐	●	●		●
LaTeX Project Public License v1.3c	●	●	●		●	●	●			●	●		●
MIT License	●	●	●		●		●				●		●
Mozilla Public License 2.0	●	●	●	●	●	●	●		◐		●	●	●
Microsoft Public License	●	●	●	●	●		●					●	●
Microsoft Reciprocal License	●	●	●	●	●	●	●		◐			●	●
University of Illinois/NCSA Open Source License	●	●	●		●		●				●		●
ODC Open Database License v1.0	●	●	●	●	●	●	●		●		●	●	●
SIL Open Font License 1.1	●	●	●		●		●		●		●		●
Open Software License 3.0	●	●	●	●	●	●	●	●	●	●	●	●	●
PostgreSQL License	●	●	●		●		●				●		●
The Unlicense	●	●	●		●						●		●
Universal Permissive License v1.0	●	●	●	●	●		●				●		●
Vim License	●	●	●		●	●	●		●	●			
Do What The F*ck You Want To Public License	●	●	●		●								
zlib License	●	●	●		●		◐			●	●		●

Figure A.1: Software licenses and their usage rights and obligations. Reproduced from (Choosealicense.com, 2019a).

B

Industry Sectors

Top level industry categories, reproduced from European Commission (2008, p.57):

- Agriculture, forestry and fishing
- Mining and quarrying
- Manufacturing
- Electricity, gas, steam and air conditioning supply
- Water supply; sewerage, waste management and remediation activities
- Construction
- Wholesale and retail trade; repair of motor vehicles and motorcycles
- Transportation and storage
- Accommodation and food service activities
- Information and communication
- Financial and insurance activities
- Real estate activities
- Professional, scientific and technical activities
- Administrative and support service activities

- Public administration and defence; compulsory social security
- Education
- Human health and social work activities
- Arts, entertainment and recreation
- Other service activities
- Activities of households as employers; undifferentiated goods- and services-producing activities of households for own use

C

Technology Factors

Table C.1: Technology Factor (TF) for languages used in applications in the case studies discussed in Section 4.4. Adapted from Jones (2017, p.49–51)

Technology	**TF (person-months)**
PL/SQL	0,5697[1]
Objective C	0,4029
Delphi	0,3900
Typescript	0,3580[2]
Ruby	0,3387
C#	0,3291
Java	0,3259
C++	0,3259
PHP	0,3259
Python	0,3259
JavaScript	0,3067
C	0,2810

[2]Value is based on "TranscriptSQL" Jones (2017, p.51)
[2]Value is based on "Mixed Languages" Jones (2017, p.51)

References

Almeida, D. A., Murphy, G. C., Wilson, G., & Hoye, M. (2017). Do Software Developers Understand Open Source Licenses? *IEEE International Conference on Program Comprehension*, 1–11. doi: 10.1109/ICPC.2017.7

Ampatzoglou, A., Ampatzoglou, A., Avgeriou, P., & Chatzigeorgiou, A. (2015). Establishing a framework for managing interest in technical debt. In *Bmsd 2015 - proceedings of the 5th international symposium on business modeling and software design* (pp. 75–85). doi: 10.5220/0005885700750085

Asnar, Y., Giorgini, P., & Mylopoulos, J. (2011). Goal-driven risk assessment in requirements engineering. *Requirements Engineering*, *16*(2), 101–116. doi: 10.1007/s00766-010-0112-x

Aven, T. (2016). Risk assessment and risk management: Review of recent advances on their foundation. *European Journal of Operational Research*, *253*(1), 1–13. doi: 10.1016/j.ejor.2015.12.023

Avgeriou, P., Kruchten, P., Ozkaya, I., & Seaman, C. (2016). Managing Technical Debt in Software Engineering (Dagstuhl Seminar 16162). *Dagstuhl Reports*, *6*(4), 110–138. doi: 10.4230/DagRep.6.4.110

Boehm, B. (2003). Value-based software engineering. *ACM SIGSOFT Software Engineering Notes*, *28*(2), 3. doi: 10.1145/638750.638775

Cailliau, A., & van Lamsweerde, A. (2013). Assessing requirements-related risks through probabilistic goals and obstacles. *Requirements Engineering*, *18*(2),

129–146. doi: 10.1007/s00766-013-0168-5

CAST. (2020a). *CAST Application Intelligence Platform.* Retrieved from https://www.castsoftware.com/products/application-intelligence-platform

CAST. (2020b). *CAST Highlight.* Retrieved from https://www.castsoftware.com/products/highlight

Choosealicense.com. (2019a). *Appendix.* Retrieved 05-03-2020, from https://choosealicense.com/appendix/

Choosealicense.com. (2019b). *No License.* Retrieved 24-02-2020, from https://choosealicense.com/no-permission/

CIRCL. (2020). *CVE Search.* Retrieved from http://cve.circl.lu/

Coelho, J., Valente, M. T., Silva, L. L., & Shihab, E. (2018). Identifying unmaintained projects in github. *International Symposium on Empirical Software Engineering and Measurement*. doi: 10.1145/3239235.3240501

Computing Technology Industry Association. (2019). *IT industry outlook 2020* (Tech. Rep.). Retrieved from https://comptiacdn.azureedge.net/webcontent/docs/default-source/research-reports/comptia-it-industry-outlook-2020.pdf?sfvrsn=8869ad68_0

Damodaran, A. (2013). *The Dark Side of Valuation* (Second ed.). New Jersey: FT Press.

De Groot, J., Nugroho, A., Bäck, T., & Visser, J. (2012). What is the value of your software? In *2012 3rd international workshop on managing technical debt, mtd 2012 - proceedings* (pp. 37–44). doi: 10.1109/MTD.2012.6225998

Dingsøyr, T., & Lassenius, C. (2016). Emerging themes in agile software de-

velopment: Introduction to the special section on continuous value delivery. *Information and Software Technology*, *77*, 56–60. doi: 10.1016/j.infsof.2016.04.018

European Commission. (2008). *NACE Rev. 2 – Statistical classification of economic activites in the European Community.*

Franch, X., Susi, A., Annosi, M. C., Ayala, C., Glott, R., Gross, D., ... Siena, A. (2013). Managing risk in open source software adoption. In *ICSOFT 2013 - proceedings of the 8th international joint conference on software technologies* (pp. 258–264). doi: 10.5220/0004592802580264

Free Software Foundation. (2019). *Frequently Asked Questions about version 2 of the GNU GPL.* Retrieved 05-03-2020, from `https://www.gnu.org/licenses/old-licenses/gpl-2.0-faq.html`

García García, J., & Alonso Magdaleno, M. (2013). Valuation of open source software: how do you put a value on free? *Revista de Gestão, Finanças e Contabilidade*, *3*(1), 3–16. doi: 10.29386/rgfc.v3i1.106

Halkjelsvik, T., & Jørgensen, M. (2018). *Time Predictions - Understanding and Avoiding Unrealism in Project Planning and Everyday Life.*

Hauge, Ø., Cruzes, D. S., Conradi, R., Velle, K. S., & Skarpenes, T. A. (2010). Risks and risk mitigation in open source software adoption: Bridging the gap between literature and practice. In *IFIP advances in information and communication technology* (Vol. 319 AICT, pp. 105–118). doi: 10.1007/978-3-642-13244-5_9

Holterman, W. (2004). Waardebepaling in het kader van de goodwill impairment-test. *Maandblad Voor Accountancy en Bedrijfseconomie*, *78*(6), 261–268. doi: 10.5117/mab.78.17510

IFRS Foundation. (2018). IFRS 13 Fair Value Measurement. In *IFRS®*

standards (pp. 667–714). Retrieved from http://eifrs.ifrs.org/eifrs/bnstandards/en/IFRS13.pdf

ISO. (2018). *ISO 31000:2018(en) Risk management — Guidelines.* Retrieved 21-02-2020, from https://www.iso.org/obp/ui/{#}iso:std:iso:31000:ed-2:v1:en

Jones, C. (1995). Backfiring: Converting lines of code to function points. *Computer*, *28*(11), 87–88. doi: 10.1109/2.471193

Jones, C. (2017). *Software Economics and Function Point Metrics: Thirty years of IFPUG Progress* (Tech. Rep.). Retrieved from https://www.ifpug.org/wp-content/uploads/2017/04/IYSM.-Thirty-years-of-IFPUG.-Software-Economics-and-Function-Point-Metrics-Capers-Jones.pdf

Kalliamvakou, E., Singer, L., Gousios, G., German, D. M., Blincoe, K., & Damian, D. (2014). *The promises and perils of mining GitHub.* doi: 10.1145/2597073.2597074

Kaplan, S., & Garrick, B. J. (1981). On The Quantitative Definition of Risk. *Risk Analysis*, *1*(1), 11–27. doi: 10.1111/j.1539-6924.1981.tb01350.x

Li, J., Conradi, R., Slyngstad, O. P., Torchiano, M., Morisio, M., & Bunse, C. (2008). A state-of-the-practice survey of risk management in development with off-the-shelf software components. *IEEE Transactions on Software Engineering*, *34*(2), 271–286. doi: 10.1109/TSE.2008.14

Linh, N. D., Hung, P. D., Diep, V. T., & Tung, T. D. (2019). Risk management in projects based on open-source software. *ACM International Conference Proceeding Series*, *Part F1479*, 178–183. doi: 10.1145/3316615.3316648

McKinsey, I., Copeland, T. E., Koller, T., & Murrin, J. (2000). *Valuation: measuring and managing the value of companies.* Wiley.

Microsoft. (2019). *Microsoft Visio.*

Nugroho, A., Visser, J., & Kuipers, T. (2011). An empirical model of technical debt and interest. In *Proceedings - international conference on software engineering* (pp. 1–8). doi: 10.1145/1985362.1985364

Pashchenko, I., Plate, H., Ponta, S. E., Sabetta, A., & Massacci, F. (2018). Vulnerable open source dependencies. In *Acm / ieee international symposium on empirical soft-ware engineering and measurement (esem) (esem '18), october 11–12, 2018, oulu, finland* (pp. 1–10). doi: 10.1145/3239235.3268920

Perl, H., Dechand, S., Smith, M., Arp, D., Yamaguchi, F., Rieck, K., ... Acar, Y. (2015). VCCFinder: Finding potential vulnerabilities in open-source projects to assist code audits. *Proceedings of the ACM Conference on Computer and Communications Security*, *2015-October*, 426–437. doi: 10.1145/2810103.2813604

Ponemon Institute. (2019). *Cost of a data breach report* (Tech. Rep.). Retrieved from `https://www.ibm.com/downloads/cas/ZBZLY7KL`

Prana, G. A. A., Treude, C., Thung, F., Atapattu, T., & Lo, D. (2019). Categorizing the Content of GitHub README Files. *Empirical Software Engineering*, *24*(3), 1296–1327. doi: 10.1007/s10664-018-9660-3

R Core Team. (2020). *R: A language and environment for statistical computing.* Vienna, Austria. Retrieved from `https://www.R-project.org`

Reilly, R. F. (2008). The Relief from Royalty Method of Intellectual Property Valuation. *Insights*, *Autumn*, 20–43. Retrieved from `http://www.willamette.com/insights_journal/08/autumn_2008_2.pdf`

Rosen, L. (2004). *Open Source Licensing.* Upper Saddle River, New Jersey: Prentice Hall.

Ruffin, M., & Ebert, C. (2004). Using Open Source Software in Product Development: A Primer. *IEEE Software*, *21*(1), 82–86. doi: 10.1109/MS.2004.1259227

Sales, T. P., Baião, F., Guizzardi, G., Almeida, J. P. A., Guarino, N., & Mylopoulos, J. (2018). The common ontology of value and risk. In *International conference on conceptual modeling* (pp. 121–135). doi: 10.1007/978-3-030-00847-5_11

Sebastiani, R., Giorgini, P., & Mylopoulos, J. (2004). Simple and minimum-cost satisfiability for goal models. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *3084*, 20–35. doi: 10.1007/978-3-540-25975-6_4

Sherlock, J., Muniswamaiah, M., Clarke, L., & Cicoria, S. (2018). Open Source Software Opportunities and Risks. *CoRR*, *abs/1812.1*, 10. Retrieved from `http://arxiv.org/abs/1812.11697`

Siena, A., Morandini, M., & Susi, A. (2014). Modelling risks in open source software component selection. In *International conference on conceptual modeling* (pp. 335–348). doi: 10.1007/978-3-319-12206-9_28

Silic, M., Back, A., & Silic, D. (2015). Taxonomy of technological risks of open source software in the enterprise adoption context. *Information and Computer Security*, *23*(5), 570–583. doi: 10.1108/ICS-08-2014-0056

Software Freedom Conservancy v. Best Buy Co., *No. 1:2009cv10155 - Document 131* (S.D.N.Y. 2010)

Software Freedom Conservancy v. Best Buy Co., *No. 1:2009cv10155 - Document 132* (S.D.N.Y. 2010)

The Free Dictionary. (n.d.). *Injunctive Relief.* Retrieved 02-03-2020, from `https://legal-dictionary.thefreedictionary.com/injunctive+relief`

Tiemann, M. (2007). *GNU Affero GPL version 3 and the "ASP loophole"*. Retrieved 05-03-2020, from `https://opensource.org/node/152`

Welte v. Fantec, *Az.: 308 O 10/13* (Landgericht Hamburg, 2010)

Yu, E. S., & Mylopoulos, J. (1994). From E-R to "A-R" - modelling strategic actor relationships for business process reengineering. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *881 LNCS*(December), 548–565. doi: 10.1007/3-540-58786-1_101

Index