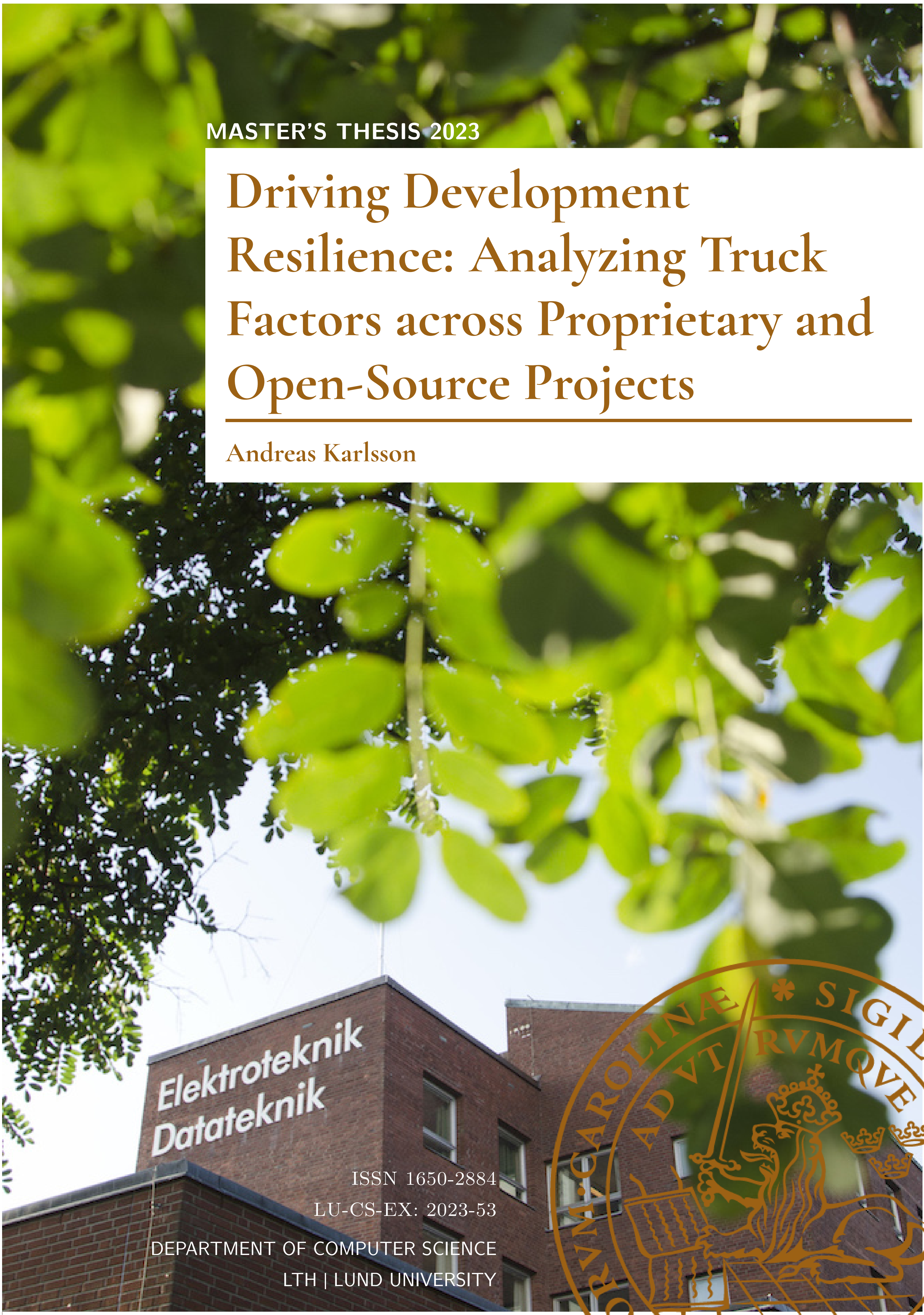


MASTER'S THESIS 2023

# Driving Development Resilience: Analyzing Truck Factors across Proprietary and Open-Source Projects

Andreas Karlsson

ISSN 1650-2884
LU-CS-EX: 2023-53

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2023-53

# Driving Development Resilience: Analyzing Truck Factors across Proprietary and Open-Source Projects

Riskmedvetenhet: Analys av Truck Faktorn i proprietära och öppna källkodsprojekt

**Andreas Karlsson**

# Driving Development Resilience: Analyzing Truck Factors across Proprietary and Open-Source Projects

Andreas Karlsson
andreasjankarlsson@gmail.com

December 19, 2023

Master's thesis work carried out at
the Department of Computer Science, Lund University.

Supervisors: Markus Borg, markus.borg@cs.lth.se
Adam Tornhill, adam.tornhill@codescene.com

Examiner: Per Runeson, per.runeson@cs.lth.se

## Abstract

[Context] The agile approach to software development has led developers to retain more project-specific knowledge. This, along with the software industry's higher turnover rate compared to other sectors, makes software projects more susceptible to the abrupt loss of key personnel. [Objective] This project investigates if the resilience to sudden loss of key developers differs in proprietary and open-source software projects. This is done through the introduction of a new algorithm, expanding upon the current state-of-the-art Truck Factor algorithm. [Method] The method comprises three phases. Phase one involves reproducing the current state-of-the-art algorithm. In phase two, we introduce five new algorithmic approaches and analyze advantageous configurations. In phase three, we implement the most promising algorithm from phase two on open-source and proprietary projects obtained from CodeScene's data lake. [Results] Two of the proposed algorithms provided more accurate results on an oracle compared to the current state-of-the-art algorithm. Furthermore, no clear difference between the Truck Factor distribution between proprietary and open-source projects could be observed. However, a trend appears for both contexts: when the number of developers increases, the relative Truck Factor decreases. [Conclusion] The execution of the most promising algorithm found that open-source and proprietary projects share similar characteristics regarding resilience to the sudden loss of key personnel.

**Keywords**: Truck Factor, Bus Factor, repository mining, software metrics, knowledge distribution, proprietary software development, open-source software development

# Acknowledgements

I would like to thank:

- **Markus Borg** for your invaluable support throughout this thesis. I am truly grateful for your assistance and guidance.
- **Adam Tornhill** for granting me the opportunity to conduct my thesis project at CodeScene. I am truly impressed by the remarkable company you have built.
- **CodeScene** for creating an inclusive environment where I felt like a part of the team.
- **Jenny Godring Gullberg** for her support and for enabling me to work at Handelsbanken alongside this master thesis.
- **Rut Montero Vilar** for her steadfast support over the last couple of years.
- **Guilherme Avelino** for his significant contributions to the field of Truck Factor algorithms and for his support during the course of this thesis.

# Contents

# Chapter 1
# Introduction

This chapter begins by addressing the necessity for a Truck Factor algorithm, followed by an exploration of the Truck Factor's definition. Subsequently, it reviews findings from prior research, examining various methods for assigning and categorizing knowledge to a file. The chapter concludes by presenting the overarching aim of this thesis.

## 1.1 Background and motivation

Historically, traditional software development was built on the idea that a successful release results from rigorous development processes. This includes intensive analysis, design and documentation phases [20]. However, with the rising challenges of the 21st century, organizations had to more quickly adapt and respond to more frequent and major changes in the business domain [6]. Traditional software development has therefore been challenged in today's world, and alternative organizational structures and development strategies have proven to provide significant business benefits [20].

Traditional software development is keen to some common points of failure, often deriving from the relationship between the business domain and software development teams. To battle these shortfalls, a more agile approach to software development has become increasingly established [2].

Traditional software development emphasizes risk reduction through comprehensive documentation and analysis, whereas agile software development seeks to mitigate risk by quickly adapting to dynamic environments [20, 32, 1]. Agile practices commonly include informal communication with stakeholders, keeping more project information within the development team (rather than in formal requirement documents), and fostering the emergence and evolution of architectures [20, 1].

These factors collectively have increased organizations' dependence on developers. Coupled with the software industry's elevated turnover rate in comparison to other sectors [21, 22], software projects are particularly vulnerable to the abrupt departure of key personnel.

This thesis aims to support organizations, both in open-source and proprietary contexts, in identifying and assessing the risks linked to dependency on a limited number of developers through the introduction of a novel algorithm. Furthermore, the thesis aims to examine whether this risk observed in open-source projects extends to proprietary projects.

## 1.2 Definition of the Truck Factor

The Truck factor (TF) is a term used in software management to define a measurement of how vulnerable a project is to a sudden loss of its key members. Specifically, it refers to the hypothetical question: "What would happen if a key member got hit by a truck?" The term accesses the risk of a project based on the number of key individuals who have critical knowledge or skills. Ricca *et al.* [23] lift that the TF could be used to quantify how well distributed the knowledge and responsibilities are among the members of a project, and an indicator of how expensive it is to replace a team member in a project [24]. A higher TF, compared to the team size, indicates a higher project resilience where the project would be more likely to function if a key member were lost. Conversely, a lower TF would indicate a higher project risk due to heavy reliance on specific individuals. The term has also been referred to as the “truck factor problem” [19], “TF developers detachment” [3] and “Bus factor” [19].

## 1.3 Previous work

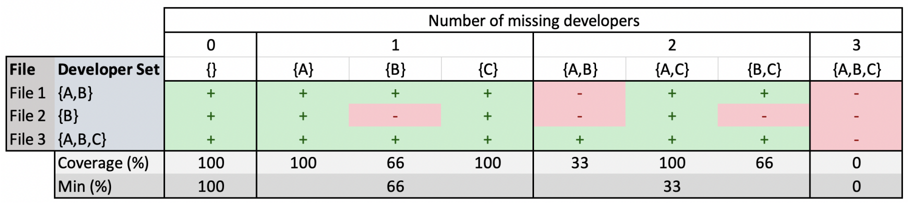
This chapter discusses the history of truck factor algorithms, different implementations of various algorithms, validations of the AVL algorithm, and definitions of file authorship.

### 1.3.1 The first TF algorithm

Zazworka *et al.* pioneered an algorithm for TF computation through mining version control system data [32]. This algorithm laid the groundwork for subsequent advancements, operating on two fundamental principles. Firstly, any developers who have done a commit to a file are assumed to possess total knowledge of the file. Secondly, a project would be at serious risk if the remaining developers' joint knowledge covers less than a threshold of files.

The algorithm functions by iteratively removing all possible sets of developers and calculating the remaining knowledge. The algorithm stops when the knowledge coverage is beyond a given threshold, and the TF is the number of iterations. An example of the algorithm is presented in Figure 1.1. With the knowledge threshold set to 70%, 50%, and 20%, the TF estimation would be 0, 1, and 2, respectively.

Ricca *et al.* conducted additional research to validate Zazworka *et al.*'s work [24]. They assessed the algorithm on 37 open-source projects with various threshold configurations. The algorithm showed potential, notwithstanding its simplistic inherent assumption that every developer who interacted with a certain file would possess equal knowledge of it. However, the algorithm showed scaling issues, especially in projects where the number of committers was greater than 30 individuals. In these projects, the computational time was measured in days rather than minutes.



|  |  | Number of missing developers |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|
|  |  | 0 | 1 |  |  | 2 |  |  | 3 |
| **File** | **Developer Set** | {} | {A} | {B} | {C} | {A,B} | {A,C} | {B,C} | {A,B,C} |
| File 1 | {A,B} | + | + | + | + | - | + | + | - |
| File 2 | {B} | + | + | - | + | - | + | - | - |
| File 3 | {A,B,C} | + | + | + | + | + | + | + | - |
|  | Coverage (%) | 100 | 100 | 66 | 100 | 33 | 100 | 66 | 0 |
|  | Min (%) | 100 | 66 |  |  | 33 |  |  | 0 |

**Figure 1.1:** An example of Zazworka *et al.*'s algorithm, plus signs imply remaining knowledge about a file. On the contrary, a minus sign implies that all knowledge of a file has been lost.

Ricca *et al.* [24] advocated that a more precise algorithm should be developed by considering that developers with a substantial number of commits are more likely to possess a greater depth of knowledge about the corresponding file, as opposed to a developer who has contributed far fewer commits.

## 1.3.2 Subsequent research

The CST algorithm presented by Cosentino *et al.* [10] builds upon Zazworka *et al.*'s algorithm. In contrast to relying on the percentage of files without knowledge, this algorithm determines the TF by considering the number of developers with sufficient understanding of the entire project to sustain development.

The algorithm estimates the TF by identifying developers with a knowledge score exceeding 100%/N, where N is the number of developers who have made a commit to the artifact. The knowledge score is derived from the average file-knowledge for a system. Cosentino *et al.* introduce four distinct metrics for computing file knowledge in a system, namely:

- *Last change takes it all:* The developer who committed the latest change is assigned to have 100% knowledge about the code in that file.
- *Multiple changes equally considered:* The knowledge about the code is assumed to be connected with the number of commits a developer has done. Thereby, the assigned knowledge per commit to a file is $\frac{100\%}{\#\text{Commits}}$
- *Non-consecutive changes:* Same as *Multiple changes equally considered*, but the knowledge about a file is split according to the number of non-consecutive commits by a user to a file.
- *Weighted non-consecutive changes:* This metric takes into account the time aspect of a commit. Commits closer in time get a higher knowledge % than commits further back in time.

The RIG algorithm developed by Riby *et al.* [25] adapts algorithms used for financial risk assessment in the financial sector to software development projects. By taking a more statistical approach, and by using the loss distribution, expected loss, knowledge at risk, and expected shortfall, Monte Carlo simulations are run with a randomly selected developer who leaves. The simulation aims to indicate how bad a loss would be, and how probable it is to occur. Through this data, the TF is determined.

The AVL algorithm proposed by Avelino *et al.* [4] follows the same principle as Zazworka *et al.*, but diverges in its methodology. Instead of assessing a commit to total knowledge of a file, AVL relies on degree-of-authorship calculations, derived from Fritz *et al.* [13], to identify one or multiple developers as authors of a file. The algorithm proposes that a project would be at serious risk if 50% of the project's files were left without one or more authors. For TF calculation, the algorithm employs a greedy approach, iteratively removing the developer responsible for the most files. The TF is determined by the number of iterations until at least 50% of the project's files lack an assigned author.

Jabrayilzade *et al.* [19] enhanced the algorithm proposed by Avelino *et al.* by incorporating additional parameters into the Degree-of-Authorship (DOA) metrics. These parameters include contribution decay, code reviews, and time spent in meetings.

The algorithms introduced by Avelino *et al.* [4], Jabrayilzade *et al.* [19], Zazworka *et al.* [32], and Cosentino *et al.* [10] treat the significance of files equally. Haratian *et al.* [16] investigated the impact of incorporating file significance into the algorithms proposed by Avelino *et al.* and Jabrayilzade *et al.* They assigned weights to files based on the number of dependencies, and found that this addition improved the performance of both algorithms.

Our literature search suggests that the field of algorithms for estimating the TF is still in its early stages, with Avelino's work being a standout contribution. Subsequent algorithms, building on Avelino *et al.*, have introduced additional metrics to improve models and better capture the knowledge within a development team. However, to the best of our knowledge, no peer-reviewed paper has utilized Lines of Code as a heuristic for assigning file-knowledge to a given developer. I.e. a large chunk of contributed source code to a file is worth, in terms of knowledge, more than a small, contrary to the existing algorithms where solely the commit-history assigns knowledge.

### 1.3.3 Validating TF algorithms

Proposing TF algorithms may be straightforward, but evaluating their accuracy presents a more challenging task. Ferreira *et al.* [12] addressed this challenge by establishing a TF oracle, consisting of 35 open-source projects. The oracle was constructed through survey-based research, building on the work conducted by Avelino *et al.* [4]. Ferreira *et al.* compared three different algorithms—AVL [4], RIG [25], and CST [10] and concluded that the AVL algorithm developed by Avelino *et al.* [4] exhibited the best accuracy. They also determined that a feasible threshold for identifying TF developers with the AVL algorithm is 50% of abandoned files. In this thesis, we use the AVL algorithm (with the 50% setting) as a benchmark for comparison. A detailed description of the AVL algorithm follows in Section 2.3.

## 1.4 Distribution of contribution in OSS

Within the Open Source community, the term *Core Developers* refers to those developers who play a prominent role in the advancement and upkeep of a software project [31]. Yamashita *et al.* [31] investigated how well Open Source development follows the Pareto principle, i.e. 20% of the developers are responsible for 80% the contributed code, by analyzing 2,496 GitHub repositories. They defined the core developers to be the group of developers accountable for the majority of contributions, encompassing up to 80% of a project's codebase. The results

**Table 1.1:** The percentage of core developers in Yamashita *et al.*'s paper [31]. Commit-based heuristics quantify core developers based on the number of commits they have made, while LoC-based heuristics rely on the number of lines of code.

| **Heuristic** | **Proportion of Core Developers** | | |
|---|---|---|---|
| | **0%-10%** | **10%-30%** | **30%-100%** |
| Commit-Based | 26% | 47% | 27% |
| LoC-based | 58% | 37% | 5% |

of the paper are accumulated in Table 1.1, and concludes that the Pareto principle is not generally applicable to open-source projects. A large portion up to a majority, depending on heuristic, of projects have a proportion of Core developers of less than 10%. Thereby, the paper indicates that open-source systems tend to have a low TF.

## 1.5 Open-source development context

Open-source software development builds on asynchronous collaboration among decentralized developers. In the early days of the development methodology, it challenged the way of building systems by demanding full transparency on both the source code, but also the governance of the project [9]. In recent years, there has been a growing focus on the open-source approach [11, 9]. Despite the absence of traditional project plans and requirements documents in open-source projects, this methodology has gained increased attention for its effectiveness in software maintenance, code reusability, and the delivery of higher-quality projects. Notably, complex systems like Linux [9] serve as a compelling example, demonstrating the success and viability of this development methodology.

A part of its initial popularity was due to the motivation of its contributors. Within Open source a driving factor is that a developer needs something specific to be done [9], and the motivation to contribute is then further reinforced when a contribution is adopted and used by other developers. This is contrary to proprietary projects, where the contributions are predefined by requirements documents.

Open-source projects have thrived under the influence of enthusiasts and amateurs for the majority of their existence [27]. However, there has been a swift adoption of these systems by companies. As of 2018, 80% of organizations rely on open-source systems [18]. Consequently, more businesses are actively engaging in open-source communities. Notably, some organizations have emerged as significant contributors, even assuming the role of principal contributors, underscoring the increasing influence of businesses in shaping and advancing open-source projects [14, 27]. Therefore, we believe that the distribution of Truck Factors between open-source and the proprietary contexts could prove to have similar characteristics.

## 1.6 Research approach

The primary goal of this thesis project is to establish the foundation for an upcoming TF algorithm to be incorporated into CodeScene's product range. This project unfolds in three key steps. Firstly, we implement various novel TF algorithms, including the state-of-the-art AVL algorithm. The validation of our AVL implementation involves reproducing analyses conducted on open-source repositories by Avelino et al. [4]. Secondly, we assess the accuracy of these algorithms using Ferreira's Oracle dataset [12]. Throughout this evaluation, we explore how fine-tuning thresholds contribute to more precise TF scores. Lastly, we apply the most accurate TF algorithms to two new datasets, one open-source and one closed-source anonymously extracted from CodeScene's customers.

Two research questions guide our discussions:

- RQ1) How do our novel TF algorithms compare to the state-of-the-art AVL algorithm?
- RQ2) To what extent does the distribution of TF of open-source GitHub projects generalize to proprietary projects?

# Chapter 2

# Background

This chapter introduces key findings from prior research and establishes definitions and heuristics that are employed throughout the thesis. Furthermore, it provides a detailed description of the AVL algorithm presented by Avelino et al. [4]. Finally, the chapter presents the dataset designated as the TF oracle.

## 2.1 Definitions

This section introduces definitions that will be used throughout the thesis. According to Avelino *et al.* [4], a file author is defined as a developer capable of maintaining a file from the most recent system snapshot onwards. This includes developers who created the file and/or those who made significant contributions to a file after its initial creation. In other words, a file could have more than one author. This thesis adopts this authorship definition.

Borg *et al.* [7] examined the correlation between developers' project experience, file ownership, and the resolution times of issues. File ownership was determined by the proportion of a developer's commits to a file, categorized on an ordinal scale: marginal ownership (<0.1), minor ownership (0.1-0.49), major ownership (0.5-0.9), and dominant ownership (>0.9). The paper concluded that current industry practices result in projects with many dominant and marginal owners, the opposite objective of collective code ownership. This thesis adopts these ownership definitions to enable a more fine-granular description of ownership on a file-level.

CodeScene is a proprietary code analysis tool that offers a palette of analyses. The tool defines a file's primary owner as the developer who has contributed at least 50% of a file's current lines of code. The CodeScene solution uses a complex implementation to track developers' contributions and ownership during the lifetime of a project. The solution is substantially more complex than the naive methods presented in previous academic tools, for example regarding tracking developer name changes and supporting pair programming configurations. In this project, we build on CodeScene's primary owner calculations to calculate TF scores as a second benchmark for comparison.

CodeScene's tools can also identify the number of active developer a project has. This data is used in the thesis to comprehend how the Truck Factor relates to the current developers of a project.

All algorithms utilized in this thesis employ greedy approaches to determine the Truck Factor. In each iteration, one developer is excluded, and consequently, if a file no longer has an assigned author, it is considered orphaned.

To summarize, we use the following definitions in this thesis:

- **Owner** A developer who has existing contributions to a file.
- **Author** A developer knowledgeable enough to maintain and further develop a file.
- **Orphaned file** A file which no longer has any authors.
- **Primary Owner** CodeScene's definition of a developer who owns 50% or more of the file's LoC.
- **Active developers** CodeScene's term for developers who made commits to the project in the last three months.

## 2.2 Heuristics for file knowledge

This section presents two fundamental heuristics in file authorship calculations. A more advanced third approach is presented together with the AVL algorithm.

### 2.2.1 Commit-based heuristic

The Commit-based heuristic assigns file-knowledge based on the number of touches a developer has done to a file, regardless of the size of the touch. The commit information can be obtained from version control systems such as *git* and be leveraged for further analyzes.

A majority of the algorithms discussed in Subsection 1.3.2 employ the Commit-based heuristic to utilize commit information for calculating the Truck Factor, each employing distinct implementation approaches.

The distinct advantages of Commit-based heuristics, when compared to LoC-based alternatives discussed in Subsection 2.2.2, lie in their consideration of a project's history rather than solely on the artefact's current state. This means that these heuristics are more resilient to refactoring activities [5]. Consequently, they can capture "hidden" knowledge that may no longer be represented in the current artifact as lines of code.

A limitation of this heuristic is that it does not consider the size or context of each commit. For instance, renaming a method in one file could trigger multiple changes in other files that call the renamed method. Consequently, the heuristic could yield a false positive contribution to files used by the previously mentioned algorithms since the commits are not weighted by context. Borg *et al.* [7] examined the file-ownership distribution in 39 proprietary systems by analyzing commit data. The results revealed that developers often modify files for which they have either minimal or dominant ownership, as illustrated in Figure 2.1. This implies that depending solely on commit-based metrics may overstate a developer's

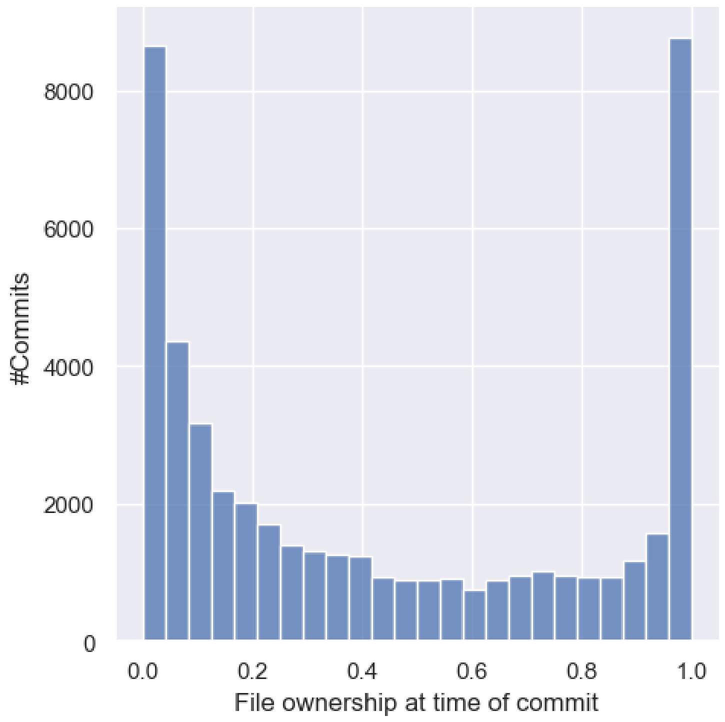


**Figure 2.1:** The ownership distribution when modifying files, adapted from Borg *et al.* [7]. The x-axis represents the relative ownership by a developer to a file, while the y-axis shows the frequency of such occurrences.

contributions. This is because having only a minimal ownership stake in a file does not necessarily demonstrate that these developers could effectively maintain and enhance the files if the main authors were to leave the project suddenly.

### 2.2.2 LoC-based heuristic

The LoC-based approach identifies a developer's contribution to a project based on the current lines of code in a given artifact.

This heuristic offers a notable advantage by placing greater emphasis on the size of each developer's contribution. It effectively mitigates the issue of false positives for developers with small ownerships, as these are overshadowed by the substantial volume of other code. However, it is crucial to acknowledge that contributions like refactoring activities by other developers can elevate the risk of suppressing existing code knowledge, potentially resulting in false negatives [5].

No Truck Factor algorithms have been presented that utilize the LoC-heuristic. Taking inspiration from Avelino et al.'s algorithm [4], this thesis defines an author in a manner akin to Avelino *et al.* Specifically, the authors of an artifact's file are the developers who own more than a set threshold of the file's lines of code. This is with the exception of the algorithm that leverages primary-author data generated from CodeScene's analysis tools. In this case, the primary author is seen as the only author of the file.

## 2.3 AVL algorithm

The AVL algorithm, developed by Avelino et al. [4], determines the Truck Factor by progressively eliminating the developer who is author of the most number of files until half of the system's files do not have an author. The algorithm does this by mapping the system's git-log to a *Degree-of-authorship* (DOA) metric for each file and developer combination. The algorithm follows five distinct phases.

**Phase I, Filter source files:** Source code files that are not of interest for the analysis should be discarded. Examples of these files are documentation, images, and third-party libraries. The algorithm does this by employing the Ruby library Linguist [1] and by filtering files explicitly listed in an ignore file.

**Phase II, handling developer aliases:** Developers may sometimes have multiple aliases due to their association with various GitHub accounts. The AVL algorithm addresses this issue in two steps. Firstly, the algorithm maps the developers' email addresses with their respective aliases. Secondly, it consolidates similar developer aliases by considering them identical if their Levenshtein distance is less than one. In other words, the algorithm considers two aliases to be the same if they differ by at most one character through insertion, deletion, or substitution. To exemplify, the aliases "developer" and "d3veloper" would be identified as the same developer since it only takes one substitution (3 $\leftrightarrow$ e) to make the aliases equal.

**Phase III, Trace change history:** The AVL algorithm traces the history of the system's files by analyzing the log history. For each change, the algorithm extracts the file's path, and the developer who committed the change, and categorizes the change into one of three categories: file addition, file modification, or file rename.

**Phase IV, Defining authors:** The AVL algorithm defines the authors of a file as the developers who have both 1) an absolute DOA greater than 3.293 and 2) a normalized DOA greater than 0.75. The DOA metric were formulated by Fritz *et al.* [13] and are determined by the functions:

$$\mathrm{DOA_{absolute}}(\mathrm{m_d}, \mathrm{f_p}) = a + b * \mathrm{FA}(m_d, f_p) + c * \mathrm{DL}(\mathrm{m_d}, \mathrm{f_p}) + d * ln(1 + \mathrm{AC}(\mathrm{m_d}, \mathrm{f_p}))$$

Where:

- $\mathrm{DOA_{absolute}}(\mathrm{m_d}, \mathrm{f_p})$, the DOA for a developer $m_d$ for file $f_p$
- *FA,First authorship* , if developer $m_d$ created the file $f_p$, then *FA*=1, else *FA*=0.
- *DL*, *Number of deliveries* how many commits the developer $m_d$ has done the a file $f_p$.
- *AC*, *Number of acceptance*, the number of commits to the file $f_p$ made by any developer other than developer $m_d$.
- **a,b,c,d**, parameters derived from Fritz *et al.* [13], Respectively: 3.293, 1.098, 0.164, -0.321.

[1]`https://github.com/github-linguist/linguist`

For each file, $f_p$, the $\mathrm{DOA}_{\mathrm{normalized}}$ is set to be 1 for the developer, $m_{MaxDOA}$, with the highest absolute $DOA_{absolute}$. Hence, the developers' $\mathrm{DOA}_{\mathrm{normalized}}$ are calculated as:

$$\mathrm{DOA}_{\mathrm{normalized}}(\mathrm{m_d}, \mathrm{f_p}) = \frac{\mathrm{DOA}_{\mathrm{absolute}}(\mathrm{m_d}, \mathrm{f_p})}{\mathrm{DOA}_{\mathrm{absolute}}(\mathrm{m_{MaxDOA}}, \mathrm{f_p})}$$

The usage of the DOA metric entails that a file can have multiple authors. Thereby taking into account scenarios where a codebase has a higher degree of knowledge distribution.

**Phase V, Estimate Truck Factor:** Using the information obtained from previous phases, the algorithm proceeds to iteratively eliminate the developer who, in each iteration, is the author of the most files. This process continues until half of the system's files no longer have any associated author. The Truck factor is the number of iterations, and the AVL algorithm outputs both the Truck Factor as well as the developers who are part of this set of developers.

Note that the Truck Factor tool has undergone updates since its first commit on July 11, 2015. The tool has three releases, v.1.0 September 12, 2015, v1.1 April 13, 2016, and the latest v.1.2 released April 12, 2018 [a].

Through email exchanges with Avelino, the distinctions between the versions were clarified. In Version 1.0, the tool did not incorporate native support for alias handling. Instead, this task was carried out manually, with the mapping added to a file that the tool later utilized as input. In subsequent releases, this functionality is already embedded in the tool. These versions also possess the capability to exclude authors associated with minor file authorships, consequently mitigating the impact of a long-tail distribution of knowledge. The algorithm achieves this by excluding developers who have authored less than 10% of the total files.

[a] `https://github.com/aserg-ufmg/Truck-Factor/releases`

## 2.4 Validation oracle

Validating the results from truck factor analyses is an acknowledged challenge. In this section, we introduce the best available oracle presented in previous work.

In Avelino et al.'s study, the AVL algorithm was applied to analyze 133 open-source systems [4]. To validate their findings, the authors conducted a survey-based investigation, consisting of 114 questionnaires tailored to individual systems, and subsequently posted them on the respective GitHub pages. The study addressed, among others, the following questions:

1. Do developers agree that the top-ranked authors are the main developers of their projects?
2. Do developers agree that their project will be in trouble if they lose the developers responding for its truck factor?

Of the 114 surveys, Avelino et al. successfully obtained responses for 62 systems. The responses to the survey's questions one and two were categorized as Agree, Partially, Disagree, and Unclear, and the results are provided in Table 2.1.

**Table 2.1:** Results of Avelino et al.[4] survey

| Question | Agree | Partially | Disagree | Unclear |
|---|---|---|---|---|
| 1 | 31 (50%) | 18 (29%) | 9 (15%) | 4 (6%) |
| 2 | 24 (39%) | 6 (10%) | 27 (43%) | 5 (8%) |

Ferreira et al. [12] subsequently conducted a comparative study to validate three different algorithms for estimating a project's Truck Factor, including the AVL algorithm. Expanding upon the surveys created by Avelino et al. [4], they introduced additional GitHub projects. Their study refined the survey responses into an oracle, comprising 35 GitHub repositories. The oracle was established based on results where the communities reached a consensus that the presented Truck Factor was correct. This dataset is regarded as the best available for benchmarking Truck Factor algorithms.

# Chapter 3
# Technical setup

This chapter presents details of the practical implementation work needed to reproduce state-of-the-art results. We also explain how we collect the datasets used in this work. Finally, we present a replication package to let others validate and build on our work.

## 3.1 State-of-the-art reproduction

As a first step, we seek to reproduce the results from the current state-of-the-art in TF calculations. This means rerunning the AVL algorithm used by Avelino *et al.* [4] and Ferreira *et al.* [12]. While an implementation of the AVL algorithm is available on GitHub, reproducing [15] the same results as in the original publications requires additional work.

To achieve this, a new shell script was developed, capable of cloning GitHub repositories and reconstructing the artifact by checking out the nearest commit following a specified date. The script performs these tasks automatically, relying on data from a given input file.

A subset of 59 GitHub repositories was cloned from Avelino et al.'s paper, selected based on the estimated Truck Factor information available on the authors' webpage[1]. Since the paper's publication in 2016, additional commits have been submitted to these repositories. Therefore, to verify their results, the version control systems need to be synchronized to the state they were in when Avelino et al. conducted their research. Contact with Avelino was initiated, and necessary files containing the date of the last commit were shared. Subsequently, the repositories are checked out to a commit on the specified days.

The artifacts from Ferreira et al.'s work [12] are reconstructed by cloning the repository and checking out commits near a specific date. The paper utilized a subset of repositories from Avelino *et al.* [4] in addition to their own research. While the paper doesn't specify the exact commit or date, it mentions checking out the subset from Avelino et al.'s repositories to a date in August 2015 and the rest in September 2016. Attempts to contact Ferreira for

[1]`http://aserg.labsoft.dcc.ufmg.br/truckfactor/survey.html`

more detailed information about the artifacts have been unsuccessful. Therefore, the subset of repositories mentioned in Avelino's paper was checked out to a commit on August 15, 2015, and the remaining repositories were checked out to a commit on September 15, 2016. Information about the repositories, GitHub links, commit-id, and dates can be found in the Appendix at Tables A.2 and A.1. Descriptions of the datasets can be found in Chapter 4.

The thesis employed two versions of the AVL algorithm: Version v1.0[2] and v1.2[3]. These versions were downloaded from GitHub, and new scripts were developed to automate the analysis of repositories. In the original research, Avelino et al. utilized the algorithm with various configuration files, managing GitHub username aliases, and excluding specific files from the analysis. These configuration files, shared by Avelino upon request, were also used in this study.

## 3.2 Data collection and analysis

Our work involves running various TF algorithms on four different datasets. We refer to the datasets as 1) TF Oracle, 2) AVL OSS, 3) CS OSS, and 4) CS Prop. The characteristics of the four datasets are described in Chapter 4. A prerequisite to run AVL is the commit-logs from the version control system. For the upcoming proposed algorithms, a completed CodeScene analysis is required.

The repositories of TF Oracle and AVL OSS were cloned and checked out locally and the commit-logs were acquired through the AVL algorithm. For the CodeScene analyses, we develop a custom script to initiate and configure the analysis for all these projects, leveraging the CodeScene on-prem server's API functionalities.

Data for both proprietary and open-source projects within CodeScene's customer portfolio, i.e CS OSS and CS Prop , are obtained through queries to CodeScene's cloud servers. This process strictly adheres to CodeScene's terms and conditions, defining the acceptable use of data to ensure compliance with privacy and business confidentiality standards. Encryption is applied to all data containing identifiable information about users, customers, files, and/or systems.

## 3.3 Replication package

To enable future replications of our work, we open-sourced a new Java project. The system follows the design in Figure 3.1. The system uses mappers to map data from the various datasets described in Chapter 4 to FileListDO, FileDO, and CSAuthorsDO objects. The source code can be found at [4].

The purpose of the class *FileDO* is to store information about a file's current authorship and the contributions of different developers in terms of lines of code. The class provides several methods, with the primary method being the ability to determine if a file has an author based on the state of the developer set in the *Developers* class and the fixed threshold in the static class *Thresholds*.

[2]`https://github.com/aserg-ufmg/Truck-Factor/releases/tag/v1.0`
[3]`https://github.com/aserg-ufmg/Truck-Factor/releases/tag/v1.2`
[4]`https://github.com/codescene-research/truck_factor`

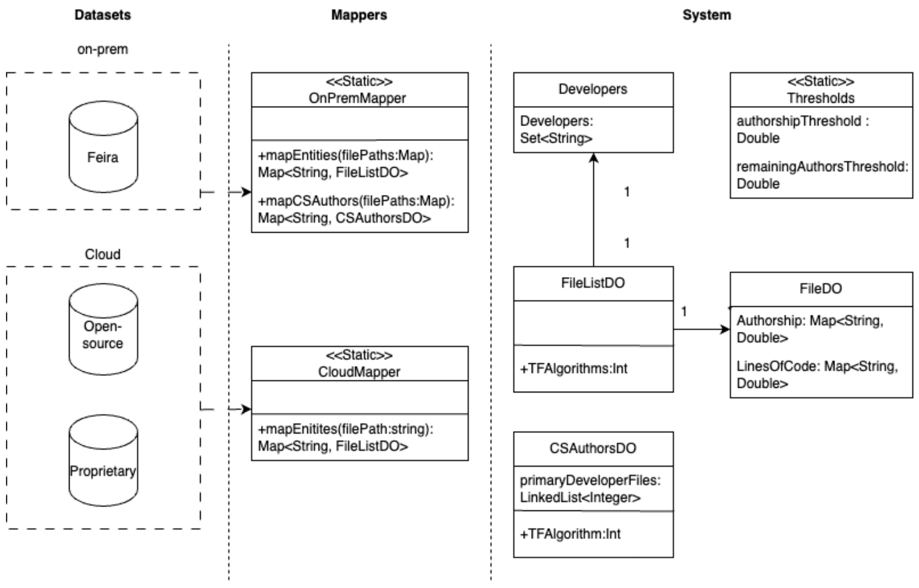

Datasets
Mappers
System
on-prem
Feira
Cloud
Open-source
Proprietary
<<Static>> OnPremMapper
+mapEntities(filePaths:Map): Map<String, FileListDO>
+mapCSAuthors(filePaths:Map): Map<String, CSAuthorsDO>
<<Static>> CloudMapper
+mapEnitites(filePath:string): Map<String, FileListDO>
Developers
Developers: Set<String>
<<Static>> Thresholds
authorshipThreshold : Double
remainingAuthorsThreshold: Double
1
1
FileListDO
+TFAlgorithms:Int
1
FileDO
Authorship: Map<String, Double>
LinesOfCode: Map<String, Double>
CSAuthorsDO
primaryDeveloperFiles: LinkedList<Integer>
+TFAlgorithm:Int

**Figure 3.1:** A simplified overview of the system's component structure.

The primary purpose of the *FileListDO* class is to store information about a project's various files and offer methods for calculating the truck factor. Additionally, the class provides methods, among others, for adding authorship to files and gathering general information about the files within the class.

The *CSAuthorDO* class is employed for estimating the Truck Factor based on primary-author data generated by CodeScene's analyses. This class utilizes a linked list to store the number of files each developer is the primary owner of. The class also supports methods for calculating the Truck Factor for the given data input.

To dynamically change the values of various thresholds, the static class Thresholds was implemented. This class is used for setting and retrieving the different thresholds that the *FileDO* and *CSAuthorsDO* classes utilize. This provides the flexibility to dynamically adjust the thresholds, as used in Section 6.2 for tuning.

# Chapter 4
# Description of datasets

This chapter provides an overview of the datasets used in the thesis. It begins by presenting the datasets, followed by a detailed description of each dataset.

## 4.1 Datasets

This chapter describes the four datasets used in this study. The term source files refers to the set of files that are part of the project's source code, i.e. files such as images, documentation, etc. are ignored.

- **TF Oracle**. 35 repositories, checked out to dates in 2015 and 2016, used by Ferreira *et al.* [12].
- **AVL OSS**. 59 repositories, checked out to dates in 2015, analyzed by Avelino *et al.* [5].
- **CS OSS**. 195 repositories collected as part of this project. The data was sampled from CodeScene's datalake. We applied a filter to only include projects with at least 100 files, 100 commits, and between 1 and 50 active developers.
- **CS Prop** 102 closed-source repositories sampled from CodeScene customers from proprietary projects. Filtered with the same strategy as CS OSS.

## 4.2 Description of datasets

In Table 4.1, descriptive statistics for the four datasets are presented. Notably, the TF Oracle and AVL OSS datasets exhibit similar mean values, with approximately 200 developers, 20 active developers, 740 source files, and 3,700 commits on average. In contrast, the CSS OS and CS Prop datasets diverge. Remarkably, the mean number of files in these datasets is

**Table 4.1:** Descriptive statistics of the datasets containing information about the datasets, with mean and standard deviation in parenthesis.

| | Oracle | AVL | CS OSS | CS Prop |
|---|---|---|---|---|
| Mean developers (StD) | 227 (335) | 193 (205) | 125 (251) | 43.2 (86) |
| Mean active developers (StD) | 19.7 (43.8) | 23.0 (34.5) | 8.5 (10.1) | 7.5 (7.5) |
| Mean source-code files (StD) | 723 (1,021) | 765 (1,032) | 1,227 (2,984) | 2,850 (16,367) |
| Mean commits (StD) | 3,757 (6,969) | 3,707 (5,217) | 3,908 (9,328) | 3,670 (6,225) |

greater, with averages of 1,227 for CS OSS and 2,850 for CS Prop. Particularly noteworthy is that the average number of active developers in these sets is approximately 8, suggesting that a relatively small number of developers are responsible for managing a larger volume of files.

In Figure 4.1, the distribution of authors across repositories in the four datasets is presented, with a logarithmic scale on the x-axis. The TF Oracle and AVL OSS datasets showcase similar characteristics, with their maximum frequency observed at around 200 developers. The CodeScene OSS dataset displays a relatively even distribution, featuring a local maximum near 120 developers. Conversely, the CodeScene Prop dataset shows a distinct peak in the [3-5] developers range, beyond which the frequency stagnates. Notably, compared to the other datasets, CodeScene Prop features a significantly lower number of developers (note the different scale on the x-axis).

Figure 4.2 outlines the distribution of active authors across repositories in the four datasets. The TF Oracle, AVL OSS, and CodeScene Prop datasets depict tendencies toward a normal distribution, AVL OSS reaching its maximum at [9-10] active developers, TF Oracle att [7-8] active developers and CodeScene Prop at [4-5] active developers. The CodeScene OSS dataset peaks at [1-2] active developers, while TF Oracle has its maximum at [4-7]. The CodeScene OSS dataset has tendencies to an exponential distribution.

In Figure 4.3, the distribution of source files for repositories in the four datasets is depicted. All datasets exhibit tendencies to follow a normal distribution, with TF Oracle peaking at 200, AVL OSS at 100, and both CodeScene OSS and CodeScene Prop. at 1,000 source files.

In Figure 4.4, the distribution of commits for repositories in the four datasets is illustrated. All datasets exhibit indications of conforming to a normal distribution, with a peak frequency of around 1,000 commits, except for CodeScene Prop, where the maximum is observed in the [2,000-3,000] commits range.

Figure 4.5 illustrates the top-10 programming languages in the datasets. Across all datasets, Java, PHP, Python, JavaScript, and C++ are consistently prominent.

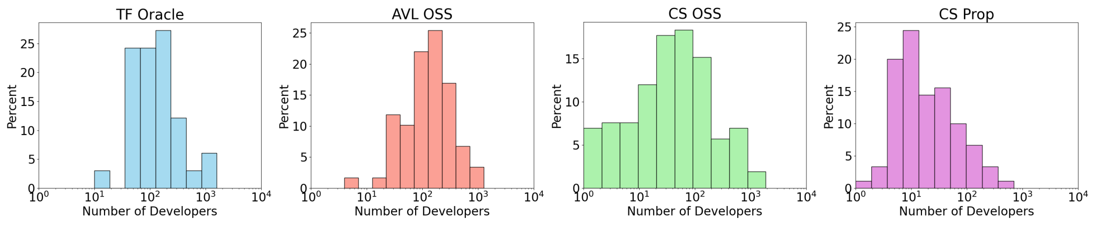


**Figure 4.1:** Distribution of developers in the four datasets.

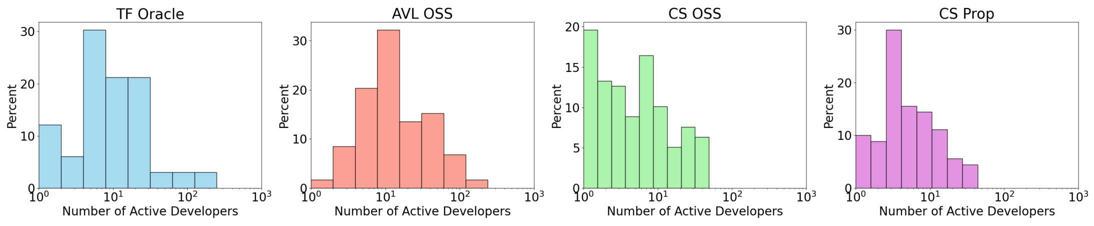


**Figure 4.2:** Distribution of active developers in the four datasets.

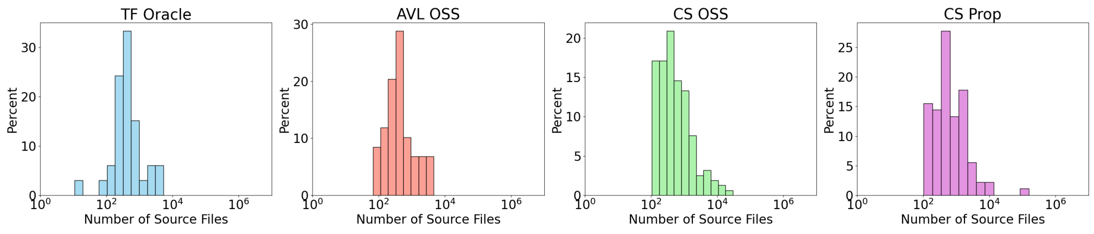


**Figure 4.3:** Distribution of files in the four datasets.

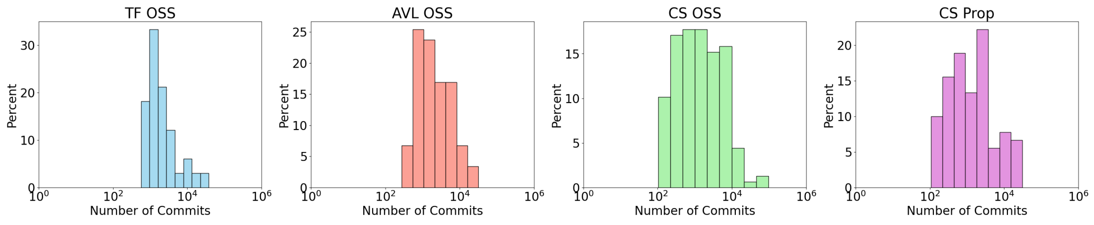
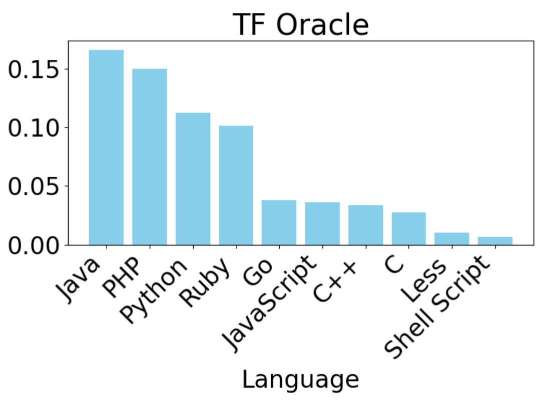
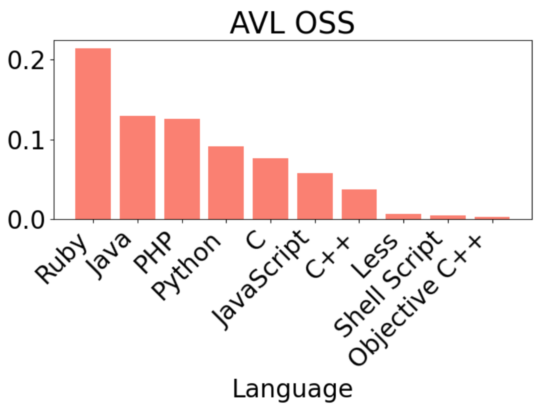


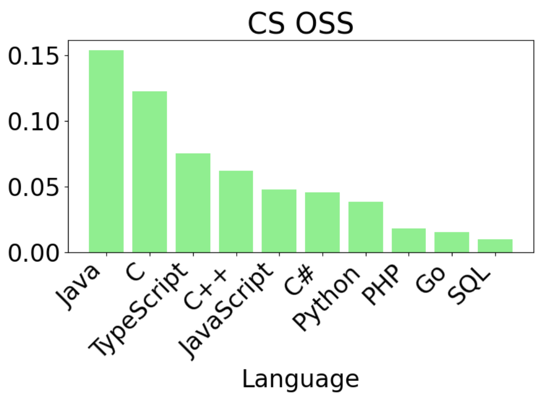
**Figure 4.4:** Distribution of commits in the four datasets.



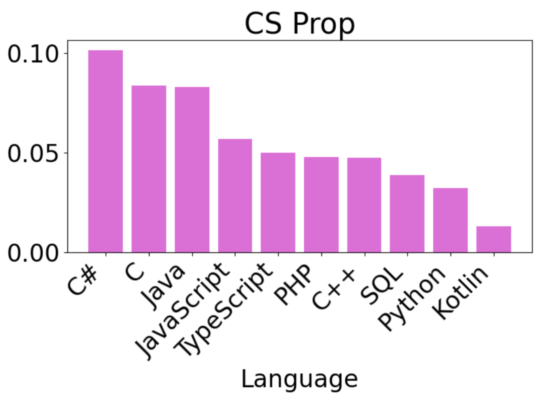
**(a)** Ferreira oracle



**(b)** Avelino



**(c)** CodeScene OSS



**(d)** CodeScene Prop

**Figure 4.5:** Top-10 programming languages in the datasets.

# Chapter 5

# Methodology

This project involves practical engineering work followed by empirical evaluations, adhering to ACM SIGSOFT Empirical Standards for Software Engineering for *benchmarking* and *repository mining*, respectively. Figure 5.1 shows an overview of the three phases in this project. The figure shows how the project uses the four different datasets described in Chapter 4.

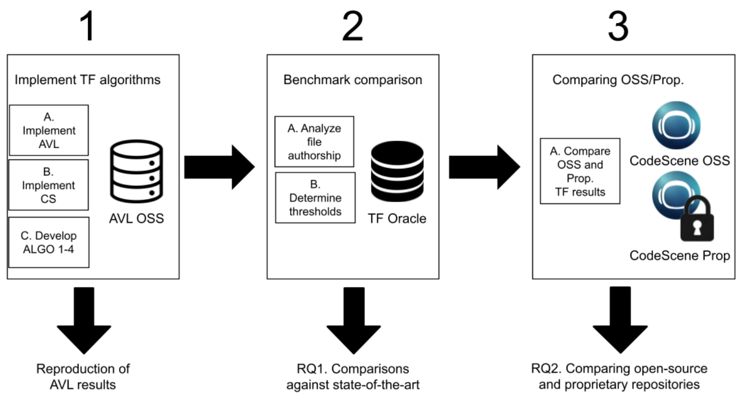

1
Implement TF algorithms
A. Implement AVL
B. Implement CS
C. Develop ALGO 1-4
AVL OSS
Reproduction of AVL results
2
Benchmark comparison
A. Analyze file authorship
B. Determine thresholds
TF Oracle
RQ1. Comparisons against state-of-the-art
3
Comparing OSS/Prop.
A. Compare OSS and Prop. TF results
CodeScene OSS
CodeScene Prop
RQ2. Comparing open-source and proprietary repositories

**Figure 5.1:** Overview of the method.

Phase 1 concerns development work to implement TF algorithms. We re-implement the AVL algorithm, develop a method for TF calculations using CodeScene's primary owner, and implement four additional TF algorithms. Using terminology proposed by Jesus *et al.*[1]. [15], we reproduce the AVL results on the AVL OSS dataset. Thus, we validate the accuracy of

[1]In this thesis, *Reproduce* is used to signify achieving results similar or equal to those presented in papers, employing the same or similar algorithms, configurations, and artifacts. In contrast, *Replicate* is employed when seeking similar or improved outcomes with new or different setups.

our implementations.

Phase 2 concerns benchmarking as described by Hasselbring [17]. A benchmark is "a study in which a software system is assessed using a standard tool (i.e. a benchmark) for competitively evaluating and comparing methods, techniques or systems according to specific characteristics such as performance, dependability, or security" [29]. In this work, we compare the six TF algorithms from Phase 1 on the OSS Oracle dataset. Also, we tune the thresholds for each algorithm to optimize the results. This threshold calibration responds to multiple calls from previous work [32, 28, 4] where thresholds were either arbitrarily set or proved challenging to establish. Output from Phase 2 provides answers for RQ1.

Phase 3 concerns repository mining [8] followed by running TF algorithms. We use CodeScene to collect one dataset of OSS repositories and another dataset for proprietary repositories. Comparing the TF scores let us answer RQ2.

## 5.1 Phase 1: Implement TF algorithms

This section details the implementation of the AVL algorithm and introduces five new alternative algorithms for determining the Truck Factor in a project. While drawing inspiration from the AVL algorithm, these proposed algorithms adopt different approaches to determine when a system is at serious risk. In contrast to the AVL algorithm, which depends on commit-based heuristics, as discussed in Subsection 2.2.1, the newly proposed algorithms employ LoC-based heuristics, as outlined in Subsection 2.2.2.

We remind the reader about our definitions in Section 2.1, i.e., any developer who has existing LoC in a file is an *owner* (marginal, minor, major, or dominant) whereas developers who are capable maintainers of the file are referred to as *authors*. In CodeScene, a developer is the *primary owner* of a file if she has contributed at least 50% of the LoC. Finally, a file is *orphaned* if it no longer has any author.

### 5.1.1 Step A: Setup AVL

The AVL algorithm is used as a benchmarking algorithm which alternative algorithms could be compared against. To ensure that this thesis uses the same Algorithm as in the paper [4], the correct installation of the algorithm is verified by running it against the AVL OSS dataset presented in Chapter 4.

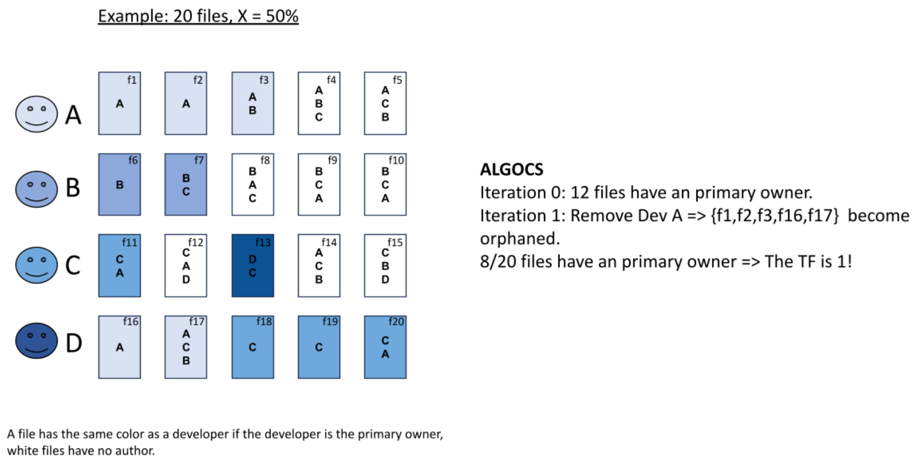
The different versions of Avelino's algorithm are each executed with two different sets of configurations. The V1.0 version is run with, and without, the configuration files provided by Avelino. The V1.2 tool is run without any configuration files and the minimum fraction of total file authorship threshold is set to Avelino's default settings, 0.00 and 0.10 respectively.

For each execution, information about each project's TF value is stored. Afterward, we compare all the results to determine which version and setup of the algorithm comes closest to reproducing the results obtained by Avelino *et al.* [4].

### 5.1.2 Step B: Setup CodeScene's approach

CodeScene currently conducts advanced analyses to determine the number of files for which the developers in a project is the primary owner. The initial proposed algorithm utilizes this information by iteratively eliminating the developer who is the primary owner of most files. The algorithm terminates when the number of files without a primary-owner falls below a predefined threshold, the Truck Factor value is determined by the number of iterations.

Figure 5.2 shows a running example of this algorithm, which we call ALGOCS . The figure illustrates an example repository containing 20 files developed by four developers (A-D), with LoC contributions reflected by the order (top to bottom). A file has the same color as a developer if the developer is the primary owner, white files have no owner. The threshold of files without an author is set to 50%.



**ALGOCS**
Iteration 0: 12 files have an primary owner.
Iteration 1: Remove Dev A => {f1,f2,f3,f16,f17} become orphaned.
8/20 files have an primary owner => The TF is 1!

A file has the same color as a developer if the developer is the primary owner, white files have no author.

**Figure 5.2:** A running example of ALGOCS

### 5.1.3 Step C: Setup alternative TF algorithms

In this subsection, we start by listing our newly proposed algorithms along with brief summaries. Then, we delve into why these algorithms were created. All these algorithms use a greedy approach to estimate the Truck Factor. In each iteration, we increase the Truck Factor by one, and the developer who is the author of the most files is removed.

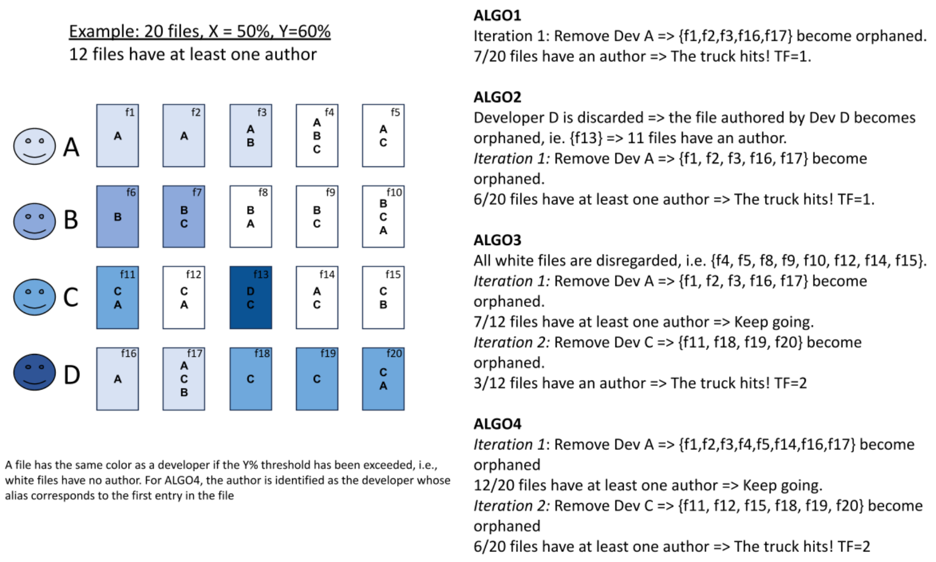
ALGO1 Developers who contributed at least X% of LoC in a file are authors. The truck hits when Y% of files are orphaned.

ALGO2 Same as ALGO1, but developers who are an author to less than 50% / #active developers of the project's total number of files are immediately discarded.

ALGO3 Same as ALGO1, but terminates when Y% of the files which initially had at least one author are orphaned.

ALGO4 The single developer who contributed the maximum number of LoC to a file is the author. The truck hits when Y% of files are orphaned.

Figure 5.3 shows a running example of ALGO 1-4. The figure illustrates an example repository containing 20 files authored by four developers (A-D). Each file has been touched by 1-4 developers, with LoC contributions reflected by the order (top to bottom). A file has the same color as a developer if the X% threshold has been exceeded, i.e., white files have no author, except for ALGO4.



Example: 20 files, X = 50%, Y=60%

12 files have at least one author

A file has the same color as a developer if the Y% threshold has been exceeded, i.e., white files have no author. For ALGO4, the author is identified as the developer whose alias corresponds to the first entry in the file

**ALGO1**

Iteration 1: Remove Dev A => {f1,f2,f3,f16,f17} become orphaned.

7/20 files have an author => The truck hits! TF=1.

**ALGO2**

Developer D is discarded => the file authored by Dev D becomes orphaned, ie. {f13} => 11 files have an author.

*Iteration 1:* Remove Dev A => {f1, f2, f3, f16, f17} become orphaned.

6/20 files have at least one author => The truck hits! TF=1.

**ALGO3**

All white files are disregarded, i.e. {f4, f5, f8, f9, f10, f12, f14, f15}.

*Iteration 1:* Remove Dev A => {f1, f2, f3, f16, f17} become orphaned.

7/12 files have at least one author => Keep going.

*Iteration 2:* Remove Dev C => {f11, f18, f19, f20} become orphaned.

3/12 files have an author => The truck hits! TF=2

**ALGO4**

*Iteration 1*: Remove Dev A => {f1,f2,f3,f4,f5,f14,f16,f17} become orphaned

12/20 files have at least one author => Keep going.

*Iteration 2:* Remove Dev C => {f11, f12, f15, f18, f19, f20} become orphaned

6/20 files have at least one author => The truck hits! TF=2

**Figure 5.3:** A running example of ALGO 1-4.

**ALGO1 :** The first proposed algorithm tries to replicate the first version of the AVL algorithm but utilizes a LoC heuristic instead of a commit-based heuristic. The algorithm defines the authors of a file as the set of developers who have contributed more than a fixed % of a file's LoC.

The algorithm iteratively removes the developer who is the author of the most files until a set % threshold of files are without an author, i.e., orphaned. The number of iterations is the truck factor estimation.

In cases with systems with a relatively high knowledge distribution, i.e., systems where many files do not have ownership exceeding the threshold of being an author, the algorithm would likely provide a Truck Factor estimation below the actual value. In extreme cases, where the knowledge distribution is equally distributed among many developers, the truck factor estimation could be zero. The AVL algorithm handles this issue by normalizing the Degree-of-Authorship. Thereby enabling several developers to be the authors of a file. To replicate this in a LoC-based heuristic would imply setting a lower threshold for being an author of a file.

Due to the logic of the algorithm, we believe that the best estimations would be given in systems where the knowledge distribution would be reasonably low. I.e., most of the files would have one single author.

**ALGO2** attempts to replicate the v1.2 version of the AVL algorithm by extending ALGO1 with the functionality to exclude developers who are authors of a relatively small number of files. In the v1.2 version of AVL, the threshold is set to 10%. However, this fixed value is likely to result in faulty truck factor estimation in projects with a large number of active developers. Hence, we have implemented a feature where ALGO2 instead discards developers who are the author of less than **50%/#*active developers*** of a project's files. We anticipate that the algorithm will share similar advantages and drawbacks as ALGO1 .

**ALGO3** : The third proposed algorithm builds on the assumption that files with a higher distribution of knowledge should affect the truck factor estimation less than those with a lower distribution. Similar to ALGO1 and ALGO2 , the authors of a file are defined as the developers who have contributed more than a specified percentage threshold of a file's LoC.

The algorithm begins by establishing the number of files with at least one author. Subsequently, it iteratively removes the developer who authored the most files in each iteration. The algorithm terminates when a specified percentage threshold of the initially identified files with at least one author becomes orphaned. The count of iterations provides the truck factor estimation.

We recognize the potential for inaccuracies in the algorithm, particularly in scenarios where file authorship is widely distributed in a project, and only a minority of the files have ownership levels surpassing the authorship threshold. In such extreme cases, the algorithm is likely to produce a truck factor estimation that is lower than expected.

**ALGO4** : The fourth algorithm takes a trivial approach to estimating the Truck Factor by assuming that a file only has one author, and the author is the developer who has contributed the most LoC.

The algorithm estimates the Truck Factor by mapping each developer to the number of files they are the author of. The algorithm then follows the logic of previously mentioned algorithms by iteratively removing the developer who is the primary owner of the greatest number of files until a given threshold of files is missing an author.

## 5.2 Phase 2: Benchmark comparison

This subchapter investigates threshold values that yield the most accurate truck factor estimations. It begins by analyzing the file ownership characteristics of software projects in both OSS and proprietary contexts. Subsequently, the algorithms are benchmarked against AVL on the TF oracle to pinpoint thresholds that result in accurate values. Ultimately, the algorithm with the thresholds that yields the best result will be selected for Phase 3.

### 5.2.1 Step A: Analyze file authorship

The distribution of file ownership is of interest in establishing reality-based thresholds for being an author of a file. Borg *et al.*'s paper [7] concluded that the number of contributions to a file, in a commit-based heuristic, follows a distinct U pattern. We aim to undertake similar investigations, exploring what patterns emerge when using LoC-based heuristics.

A new Python script was developed to visualize authorship distribution, utilizing both the OSS and proprietary datasets queried from CodeScene's servers. The script generates bar plots for each file, depicting authorship percentages. Five graphs were created for each dataset to explore the impact of the number of active developers on authorship characteristics.

Subsequently, the script is extended to plot the maximum authorship value for each file. Again, five graphs were constructed for each dataset to analyze the influence of the number of active developers on these maximum authorship values.

### 5.2.2 Step B: Determine thresholds

The insights provided in Subsection 5.2.1 suggest that a meaningful threshold for considering a developer as an author of a file falls within the range of 20% to 100%. To more precisely identify this threshold and determine the threshold for how many files can be orphaned before the truck hits, an investigation into the performance of the proposed algorithms with various thresholds was conducted.

To evaluate the impact of different thresholds on the outcomes of the proposed algorithms, a tuning suite was implemented into the Java project. For each algorithm, a grid search[2] was performed to provide *fairness* [17] for all configuration combination, thereby all possible threshold settings were executed. In each iteration, key metrics such as the number of correctly estimated Truck Factor values, the mean absolute Truck Factor differences, and the mean squared differences compared to Ferreira's oracle were recorded and stored. These metrics were then used to compare the algorithms' performance compared to the ones generated by AVL.

ALGO1 , ALGO2 , and ALGO3 were run 10,000 times each. In each run, the thresholds for orphaned files and the LoC threshold for a developer to qualify as an author were adjusted. The adjustment ranged from 0.00 to 1.00 with each increment being 0.01.

ALGO4 and ALGOCS were evaluated in a similar manner. Since the algorithms only depend on the threshold of orphaned files, 100 iterations were executed where the thresholds range from 0.00 to 1.00 and each increment was 0.01.

To ensure the benchmark *relevance* and *verifiability*, the measurements obtained were leveraged to identify regions of interest through the creation of heatmaps for ALGO1, ALGO2, and ALGO3. ALGOCS and ALGO4 were plotted to line graphs. To address the influence of noise in the measurements, an averaging technique was applied. Each point in the grid search was averaged with nearby points within an equidistant distance of less than 0.02 units. This averaging strategy aimed to mitigate variations and enhance the clarity of trends in the data.

### 5.2.3 Step C: Compare TF algorithms

The heatmaps and line graphs produced in Subsection 5.2.2 were examined to identify regions of interest, reflecting thresholds where accurate TF estimations were consistently observed. Once identified, the threshold settings and the performance metrics for all algorithms were compiled into a table.

The performances of the algorithms were compared, and an evaluation was conducted to determine which algorithm to use in Phase 3. The selection of the algorithm was based on considerations such as performance correctness, mean absolute error, and mean squared error.

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

## 5.3 Phase 3: Comparing open-source and proprietary repositories

This subchapter examines the representation of the Truck Factor distribution in proprietary and open-source projects. By utilizing the CS Prop and CS OSS datasets.

### 5.3.1 A: Compare TF results

To characterize the distribution of the Truck Factor in the context of modern development, encompassing various organizational sizes and language usage, the 195 open-source repositories in the CS OSS dataset and the 102 proprietary repositories in CS Prop were subjected to analysis using ALGO1. The criterion 'to be an author of a file' was set to 30%, while remaining files without an author' was set to 50%. These specific thresholds were determined to yield the most reliable Truck Factor estimations, as detailed in Subsection 6.2.3.

The results were collected and processed, culminating in the generation of six graphs. These graphs depict the frequency of TF values, the frequency of relative TF obtained by dividing the absolute TF by the project's number of active developers, and two box-and-whisker plots illustrating the relative and absolute TF values in comparison to the number of active developers.

# Chapter 6
# Results

This chapter unveils the results obtained from the three phases outlined in the methodology chapter. Firstly, it presents the findings from the AVL replication; secondly, it delves into authorship on a file-level. Following that, the results of the benchmarking process for the proposed algorithms are discussed, culminating in the selection of the most promising algorithm. Subsequently, this chosen algorithm is evaluated on both open-source and proprietary repositories extracted from CodeScene's customers.

## 6.1 Phase 1: Setup AVL algorithm

The results of the *reproduction* [15] procedure for the installation of AVL are presented in Table 6.1. For some repositories, the Truck Factor differs by one developer. However, in all cases, the list of developers included in the Truck Factor calculations was a subset of the developers from Avelino's data. The differences are believed to originate from slightly different configurations for the algorithm. When Avelino et al. conducted their research, they utilized the v.1.0 version of the algorithm along with specific configuration files designed for managing ignored files and alias handling. However, due to the elapsed time since the completion of Avelino *et al.*'s work, it is possible that we have slightly different files.

For the repositories Faker, monolog, and Homebrew-cask the different setups had the greatest differences. For Faker and monolog, the Truck Factor was correct for all algorithms except v.1.2 with the threshold of minimum owned files set at 0.10. Judging by the responses of Avelino's surveys [1][2], the value of the v.1.2 version with the 0.1 setting seems to be a more correct estimation of the Truck Factor.

For the repository resque, all of the instances resulted in the wrong Truck Factor value. In addition, the output of the algorithms assigned a single developer (Chris W.) to be the author

[1]`https://github.com/fzaninotto/Faker/issues/656`
[2]`https://github.com/Seldaek/monolog/issues/626`

of 61,67% of the systems files, instead of 23% stated in Issue 1317[3]. The underlying reason for this has not been identified.

For Homebrew-cask the v.1.2 algorithm with the min-threshold setting set to 0.10 provided the best estimates. We believe that when Avelino conducted his research, he manually added files which would be ignored, and hence, a lower Truck Factor was achieved. The files handed over by Avelino did not have any information about Homebrew-cask. Hence, we believe that these errors derive from missing information in the ignore-files file. Since the v.1.2 algorithm has the ability to filter out developers whose contribution is less than a threshold of the total files, small contributions are ignored. Judging by these results, it appears that the threshold minimizes the need to manually set up which files of a system should be ignored.

Of all of the versions, v.1.2 with the min-threshold set to 10% provided the results closest to Avelino *et al* [4]. Hence, this version and configuration will be the algorithm we will continue using for the remainder of this thesis.

**Table 6.1:** The different AVL versions' results on Avelino's validation suite. Results are presented as recorded TF with the difference from Avelino *et al.*[4] in parantesis. Each column represents a certain configuration of the algorithm. The *v.1.0-with files* were run with configuration files retrieved by contact with Avelino. The *min-threshold* configuration discards developers who are authors to fewer than a % of the repository's total files.

| **Repository** | **TF** | **v.1.0: no-files** | **v.1.0: with files** | **v.1.2: min-threshold = 0.1** | **v.1.2: min-threshold = 0.0** |
|---|---|---|---|---|---|
| fzaninotto/Faker | 23 | 23(0) | 23(0) | 5(-18) | 23(0) |
| fog/fog | 12 | 12(0) | 12(0) | 12(0) | 12(0) |
| saltstack/salt | 11 | 11(0) | 11(0) | 11(0) | 11(0) |
| Seldaek/monolog | 11 | 11(0) | 11(0) | 2(-9) | 11(0) |
| joomla/joomla-cms | 7 | 7(0) | 7(0) | 7(0) | 7(0) |
| scikit-learn/scikit-learn | 7 | 8(1) | 8(1) | 8(1) | 8(1) |
| chef/chef | 6 | 5(-1) | 5(-1) | 5(-1) | 5(-1) |
| emberjs/ember.js | 6 | 5(-1) | 5(-1) | 5(-1) | 5(-1) |
| resque/resque | 6 | 1(-5) | 1(-5) | 1(-5) | 1(-5) |
| spotify/luigi | 6 | 6(0) | 6(0) | 6(0) | 6(0) |
| ipython/ipython | 4 | 4(0) | 4(0) | 4(0) | 4(0) |
| jquery/jquery | 4 | 4(0) | 4(0) | 4(0) | 4(0) |
| bitcoin/bitcoin | 3 | 3(0) | 3(0) | 4(1) | 4(1) |
| yiisoft/yii2 | 3 | 0(-3) | 0(-3) | 3(0) | 3(0) |
| clojure/clojure | 2 | 2(0) | 2(0) | 2(0) | 2(0) |
| composer/composer | 2 | 2(0) | 2(0) | 2(0) | 2(0) |
| elasticsearch/elasticsearch | 2 | 2(0) | 2(0) | 2(0) | 2(0) |
| elasticsearch/logstash | 2 | 2(0) | 2(0) | 2(0) | 2(0) |

[3]`https://github.com/resque/resque/issues/1317`

| | | | | | |
|---|---|---|---|---|---|
| excilys/androidannotations | 2 | 2(0) | 2(0) | 2(0) | 2(0) |
| facebook/osquery | 2 | 2(0) | 2(0) | 2(0) | 2(0) |
| FriendsOfPHP/PHP-CS-Fixer | 2 | 2(0) | 2(0) | 2(0) | 2(0) |
| github/linguist | 2 | 2(0) | 2(0) | 2(0) | 2(0) |
| jadejs/jade | 2 | 2(0) | 2(0) | 2(0) | 2(0) |
| JohnLangford/vowpal_wabbit | 2 | 2(0) | 2(0) | 2(0) | 2(0) |
| libgdx/libgdx | 2 | 2(0) | 2(0) | 2(0) | 2(0) |
| meskyanichi/backup | 2 | 2(0) | 2(0) | 2(0) | 2(0) |
| netty/netty | 2 | 2(0) | 2(0) | 2(0) | 2(0) |
| openframeworks/openFrameworks | 2 | 2(0) | 2(0) | 2(0) | 2(0) |
| Respect/Validation | 2 | 2(0) | 2(0) | 2(0) | 2(0) |
| sampsyo/beets | 2 | 3(1) | 3(1) | 3(1) | 3(1) |
| SFTtech/openage | 2 | 2(0) | 2(0) | 2(0) | 2(0) |
| strongloop/express | 2 | 1(-1) | 1(-1) | 1(-1) | 1(-1) |
| xetorthio/jedis | 2 | 2(0) | 2(0) | 2(0) | 2(0) |
| atom/atom-shell | 1 | 1(0) | 1(0) | 1(0) | 1(0) |
| bjorn/tiled | 1 | 1(0) | 1(0) | 1(0) | 1(0) |
| bumptech/glide | 1 | 0(-1) | 0(-1) | 0(-1) | 0(-1) |
| caskroom/homebrew-cask | 1 | 53(52) | 53(52) | 1(0) | 53(52) |
| celluloid/celluloid | 1 | 1(0) | 1(0) | 1(0) | 1(0) |
| dropwizard/dropwizard | 1 | 1(0) | 1(0) | 1(0) | 1(0) |
| dropwizard/metrics | 1 | 1(0) | 1(0) | 1(0) | 1(0) |
| getsentry/sentry | 1 | 1(0) | 1(0) | 1(0) | 1(0) |
| github/android | 1 | 1(0) | 1(0) | 1(0) | 1(0) |
| gruntjs/grunt | 1 | 1(0) | 1(0) | 1(0) | 1(0) |
| janl/mustache.js | 1 | 1(0) | 1(0) | 1(0) | 1(0) |
| jrburke/requirejs | 1 | 1(0) | 1(0) | 1(0) | 1(0) |
| justinfrench/formtastic | 1 | 1(0) | 1(0) | 1(0) | 1(0) |
| kivy/kivy | 1 | 1(0) | 1(0) | 1(0) | 1(0) |
| Leaflet/Leaflet | 1 | 1(0) | 1(0) | 1(0) | 1(0) |
| less/less.js | 1 | 2(1) | 2(1) | 2(1) | 2(1) |
| mailpile/Mailpile | 1 | 2(1) | 2(1) | 2(1) | 2(1) |
| mbostock/d3 | 1 | 1(0) | 1(0) | 1(0) | 1(0) |
| powerline/powerline | 1 | 1(0) | 1(0) | 1(0) | 1(0) |
| puphpet/puphpet | 1 | 0(-1) | 0(-1) | 0(-1) | 0(-1) |
| ratchetphp/Ratchet | 1 | 1(0) | 1(0) | 1(0) | 1(0) |
| ReactiveX/RxJava | 1 | 1(0) | 1(0) | 1(0) | 1(0) |
| sandstorm-io/capnproto | 1 | 1(0) | 1(0) | 1(0) | 1(0) |
| sebastianbergmann/phpunit | 1 | 1(0) | 1(0) | 1(0) | 1(0) |
| silexphp/Silex | 1 | 1(0) | 1(0) | 1(0) | 1(0) |
| thoughtbot/paperclip | 1 | 0(-1) | 0(-1) | 1(0) | 1(0) |
| wp-cli/wp-cli | 1 | 1(0) | 1(0) | 1(0) | 1(0) |
| **Tot correct** | | 47 | 47 | 47 | 48 |
| **Percentage correct** | | 78% | 78% | 78% | 80% |

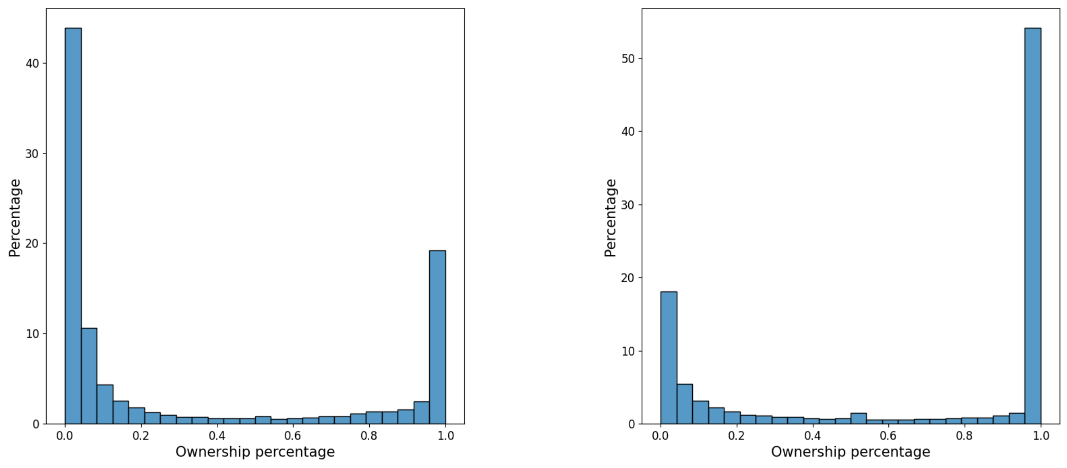
| | | | | | |
|---|---|---|---|---|---|
| **Mean Absolute diff** | | 1,17 | 1,17 | 0,70 | 1,12 |
| **Mean Squared diff** | | 45,8 | 45,8 | 7,33 | 45,65 |

## 6.2 Phase 2: Identify thresholds and a favourable algorithm

This section lifts findings regarding the distribution of file authorship and the threshold's effect on the algorithms' performances.

### 6.2.1 File authorship

The distribution of file ownership displays a distinct U-shaped pattern in both the OSS and proprietary datasets. However, in OSS projects, it is more common for files to have a larger number of smaller contributions, while in proprietary systems, it is more prevalent for files to have larger contributions as seen in Figure 6.1.



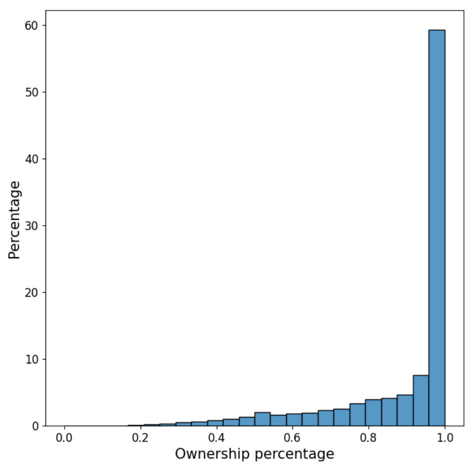
**(a)** Open-source: 1-50 active developers

**(b)** Proprietary: 1-50 active developers

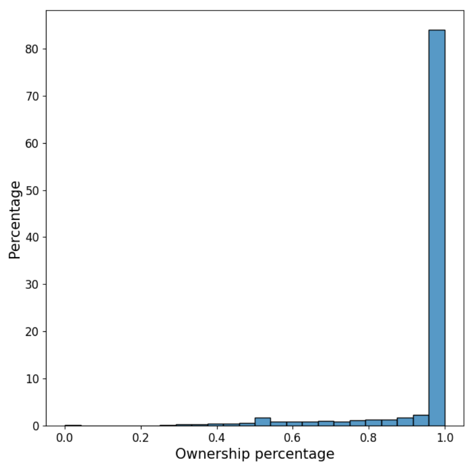
**Figure 6.1:** The distribution of file ownership in the OSS and proprietary datasets queried from CodeScene

The distribution of maximum file ownership indicates that files are commonly developed by a single developer, particularly in proprietary systems, as illustrated in Figure 6.2. Although the distribution is wider in OSS projects, it is rare for a file to have a maximum ownership of less than 50%.

These findings suggest that the majority of a file's LoC is typically written by a single developer. However, the distribution also reveals a portion of files with an ownership between 0.20 and 0.50. This suggests that developers contributing within or greater than this range could potentially be considered as part of the set of authors for a given file. This is



**(a)** Open-source: 1-50 active developers



**(b)** Proprietary: 1-50 active developers

**Figure 6.2:** The distribution of maximum file ownership in the OSS and proprietary datasets queried from CodeScene.

grounded in the assumption that developers contributing within this ownership range are, at the very least, familiar with the file. Graphs containing more information about the various distributions with higher granularity can be found in the Appendix at Figures B.1 and B.2.

### 6.2.2 Benchmarking

In this Subsection, the heatmaps with accumulated results presents thresholds which entails better metrics than AVL. The mean squared error is visualized in green, the mean absolute difference in red, and the percentage of correct TF estimations in blue. The subsection uses the convention (Fraction of LoC to be a file author, Fraction of authored files when truck hits) to describe the positions for the heatmaps' results.

The AVL algorithm's results serve as a benchmark for evaluating the proposed algorithms. AVL estimated the truck factor correctly for 19 of the 35 (54%) repositories in the TF Oracle, with a mean absolute difference of 0.86 and a mean squared difference of 4.23.

ALGO1's results from the grid search are visualized in Figures 6.5 and 6.3. The heatmap visualized in Figure 6.3 hints two regions witch configuration settings could entail a better performance than AVL on all recorded metrics. These regions are identified to be in the proximity (0.30,0.50) and (0.75,0.29). Detailed metrics for these points are provided in Table 6.2.

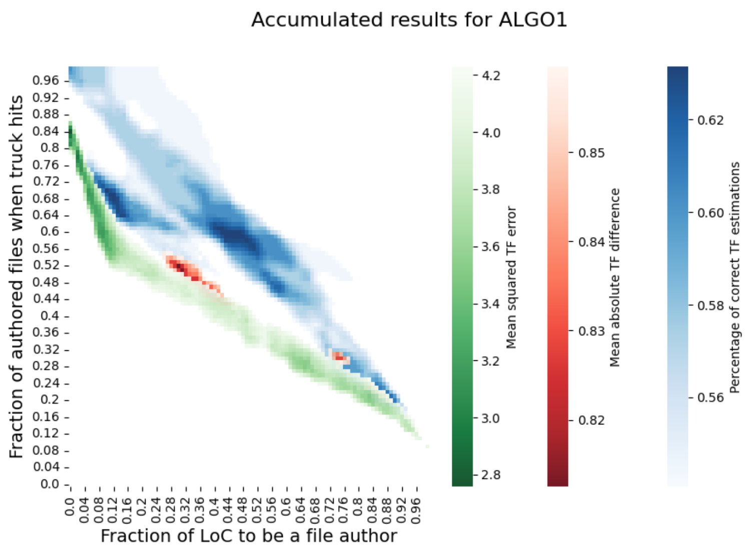


**Figure 6.3:** Threshold configurations for which ALGO1 outperforms AVL. Regions where the three colors overlap are deemed to be of the most interest.

ALGO2's threshold analysis are presented in Figure 6.5 and 6.4. The algorithm does not have any regions that provide a lower Mean absolute error or Mean squared error than AVL. However, the algorithm presents a greater percentage of correctly estimated TF values compared to AVL, at the region in the proximity of (0.64,0.34).

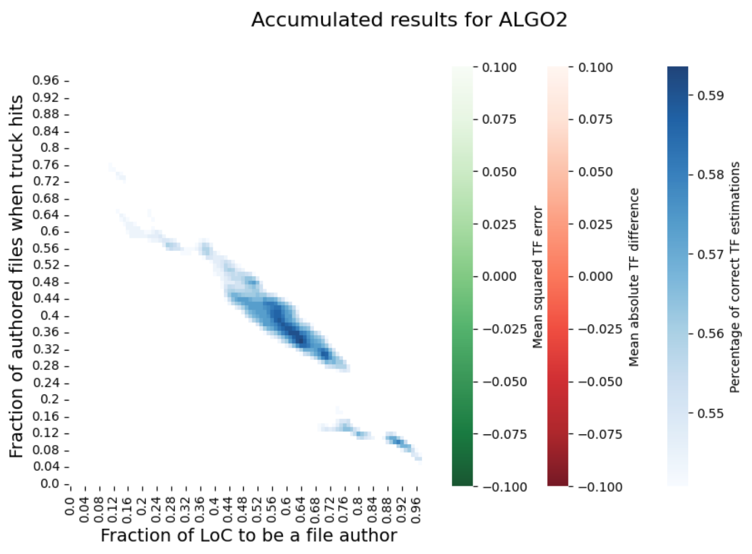
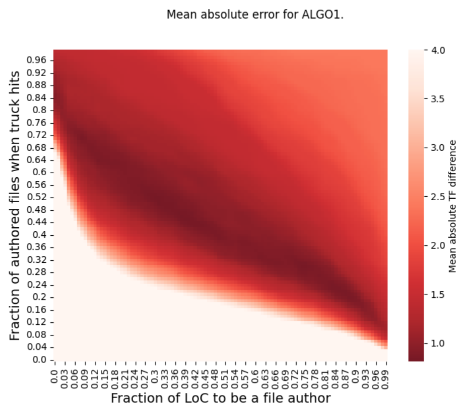
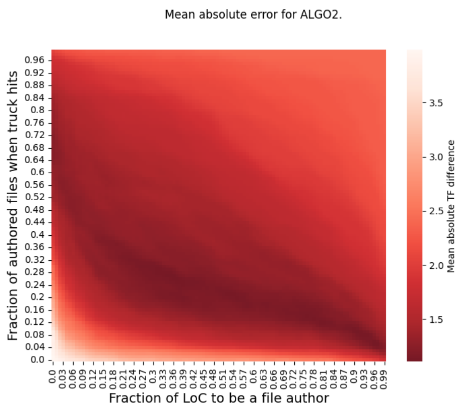


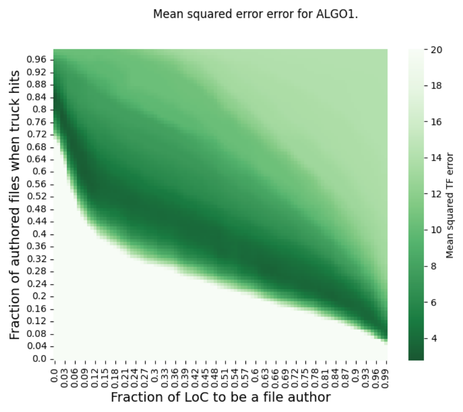
**Figure 6.4:** Threshold configurations for which ALGO2 outperforms AVL on the recorded metrics.



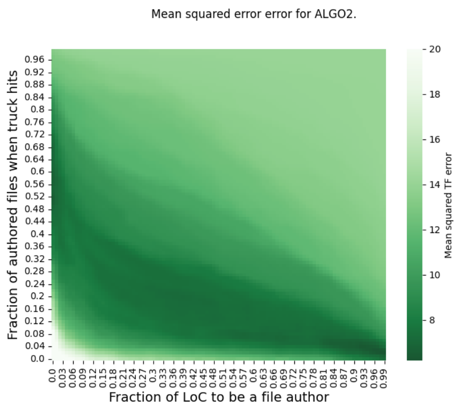
**(a)** ALGO1 : MAE.



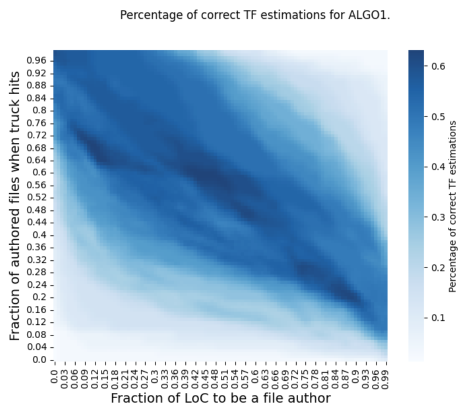
**(b)** ALGO2 : MAE.



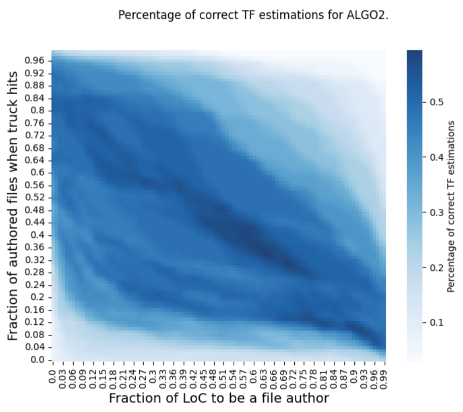
**(c)** ALGO1 : MSE.



**(d)** ALGO2 : MSE.



**(e)** ALGO1 : Mean correct %



**(f)** ALGO2 : Mean correct %

**Figure 6.5:** Results of the ALGO1 and ALGO2 threshold analysis. In the figures for mean absolute truck factor errors and mean squared truck factor error, the values have been capped at 4.0 and 20.0, respectively, for visualization purposes.

The analysis results of ALGO3 are presented in Figures 6.7 and 6.6. The two regions in proximity to (0.36,0.52) and (0.70,0.49) are of interest. This due to the overlapping in the heatmap at Figure 6.6

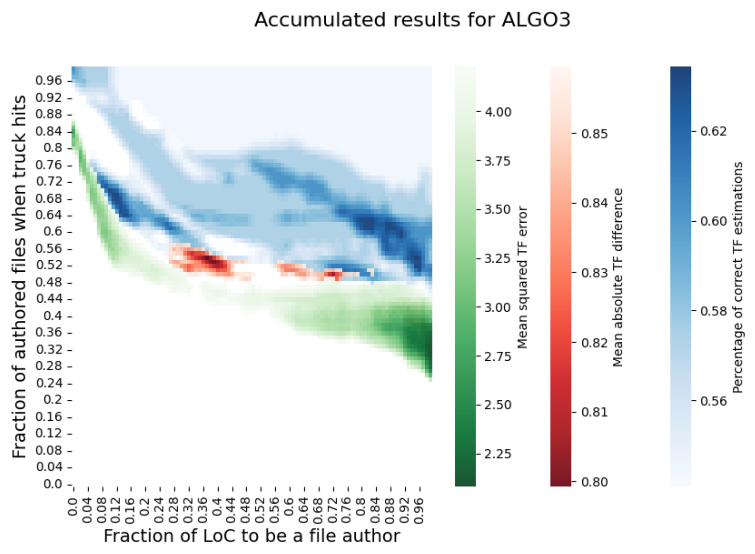
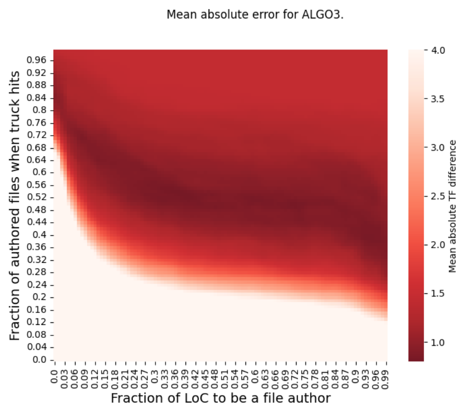
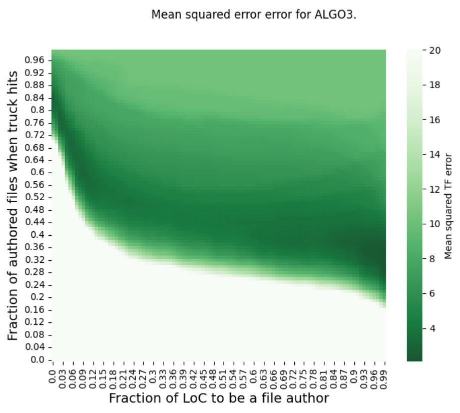


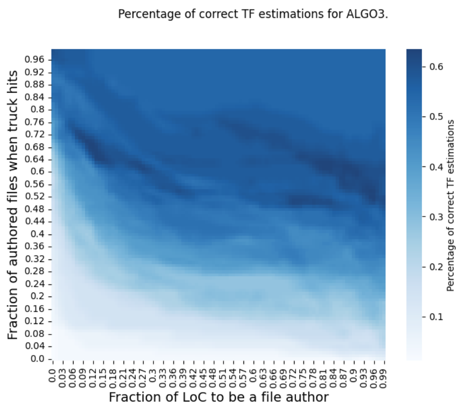
**Figure 6.6:** Threshold configurations for which ALGO3 outperforms AVL.



**(a)** ALGO3 : MAE.



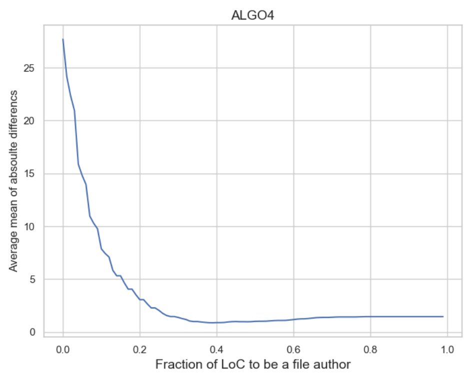
**(b)** ALGO3 : MSE.



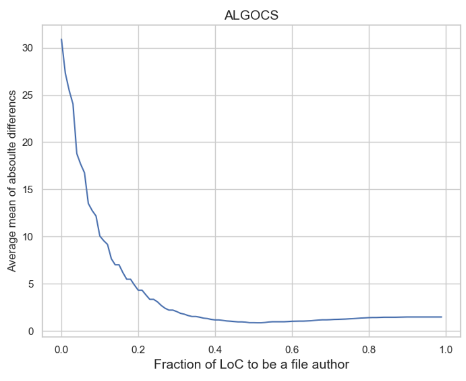
**(c)** ALGO3 : Mean correct %.

**Figure 6.7:** Results of ALGO3 threshold analysis. In the figures for mean absolute truck factor difference and mean squared truck factor error, the values have been capped at 4.0 and 20.0, respectively, for visualization purposes.

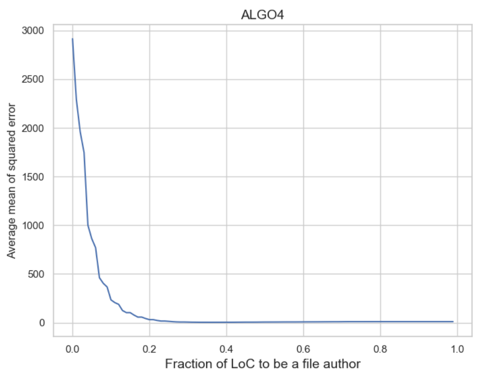
The results from ALGO4 are presented in Figure 6.8. The most beneficiary threshold configurations for each evaluations metric are (0.36),(0.39), and (0.58), respectively. More detailed results are presented in Table 6.2.



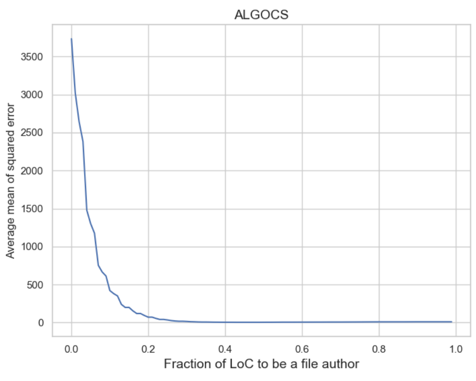
**(a)** ALGO4 : MAE.



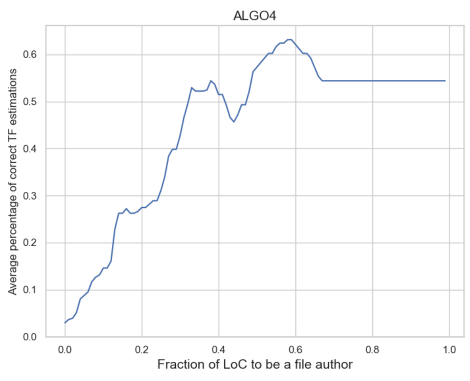
**(b)** ALGOCS : MAE.



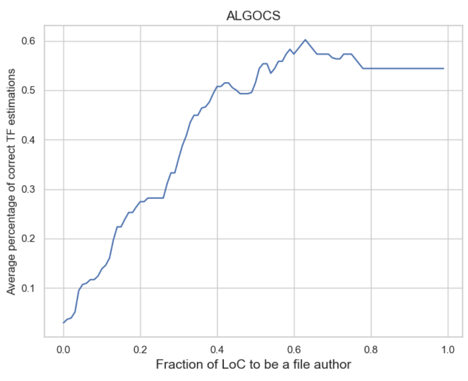
**(c)** ALGO4 : MSE.



**(d)** ALGOCS : MSE.



**(e)** ALGO4 : Mean correct %.



**(f)** ALGOCS : Mean correct %.

**Figure 6.8:** Results of ALGO4 and ALGOCS threshold analysis. The figures display the Mean Absolute Error (MAE), Mean Square Error (MSE), and the percentage of correct TF estimations. Depending on the threshold of remaining files with at least one author.

As shown in Figure 6.8, ALGOCS 's most favorable results are identified at the threshold configurations at the points (0.63),(0.52), and (0.44), respectively. More details of the metrics are presented in Table 6.2

### 6.2.3 Summary thresholds

**Table 6.2:** Accuracy of the five algorithms on the oracle dataset. Each value is averaged by the points in proximity of 0.02 Euclidian distances. Cells highlighted in green have a better result than AVL.

| Algorithm | Position | Eukledian average | | |
|---|---|---|---|---|
| | | % Correct | Mean absolute error | Mean square error |
| AVL | - | 0.54 | 0.86 | 4.23 |
| ALGO1 | (0.30,0.50) | 0.55 | 0.85 | 4.17 |
| ALGO1 | (0.75,0.29) | 0.61 | 0.84 | 4.23 |
| ALGO2 | (0.64,0.34) | 0.59 | 1.33 | 9.96 |
| ALGO3 | (0.36,0.52) | 0.54 | 0.83 | 4.72 |
| ALGO3 | (0.70,0.49) | 0.57 | 0.83 | 4.21 |
| ALGO4 | (_._,0.36) | 0.52 | 0.95 | 3.76 |
| ALGO4 | (_._,0.39) | 0.54 | 0.88 | 4.00 |
| ALGO4 | (_._,0.58) | 0.63 | 1.10 | 7,24 |
| ALGOCS | (_._,0.63) | 0.60 | 1.05 | 6.72 |
| ALGOCS | (_._,0.52) | 0.55 | 0.87 | 4.93 |
| ALGOCS | (_._,0.44) | 0.50 | 1.04 | 4.37 |

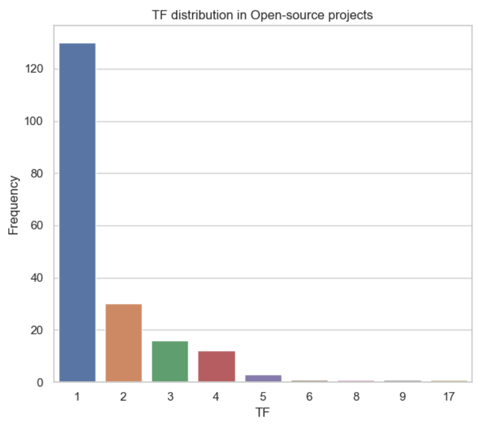
The results in Figure 6.2 suggest that practical thresholds for all of the proposed algorithms are achievable. For all of the algorithms, except ALGO2 , the threshold for orphaned files before the truck hits is in the interval [29%, 50%], which is supported in our findings presented in Subsection 6.2.1. This threshold interval also includes the threshold set by Avelino *et al.*[4].

The accumulated results of the threshold analysis are shown in Table 6.2. ALGO1 and ALGO3 had configuration settings which resulted more favourable TF metrics compared to AVL. Both of the algorithms presented similar results. however, of the two algorithms, ALGO1 , with the settings (0.30,0.50) was deemed to be the most feasible algorithm for Phase 3. This decision was founded on the findings in Subsection 6.2.1, and for its lower Mean square error.

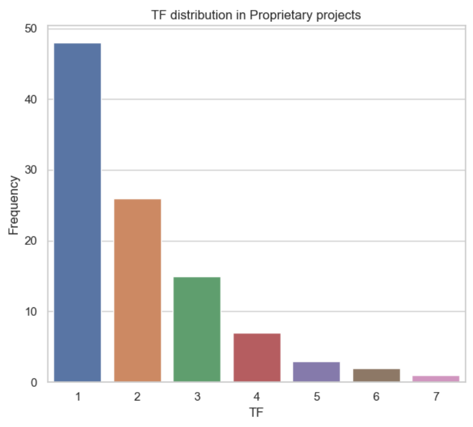
As discussed in Section 2.1, Avelino *et al.* [4] defines an author as a developer who is capable of maintaining a file from the most recent system snapshot onwards. ALGO1's authorship-threshold of 0.30 indicates that this definitely is the case.

## 6.3 Phase 3: Comparing open-source and proprietary repositories

The distribution of Truck Factors for the two datasets are are displayed in Figure 6.9. Both CS OSS and CSS Prop demonstrate a right-skewed TF distribution where the open-source dataset's tail is comparatively less pronounced. The distribution indicates that projects generally tend to have a low truck factor. It is noteworthy that the distribution of TF results does not align with the distributions of active developers, as presented in Chapter 4. This suggests that, overall, the TF is generally much lower than the number of active developers.



**(a)** CS OSS



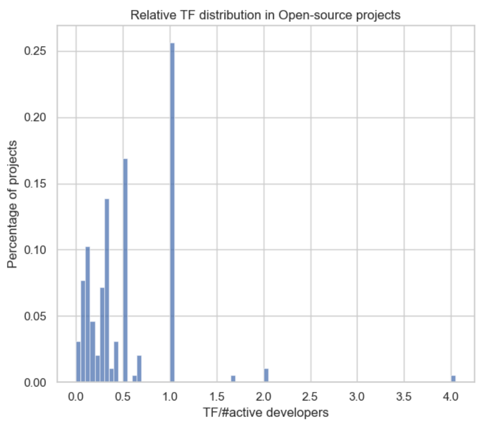
**(b)** CS Prop

**Figure 6.9:** The frequency of TF values in Proprietary and Open-source projects.

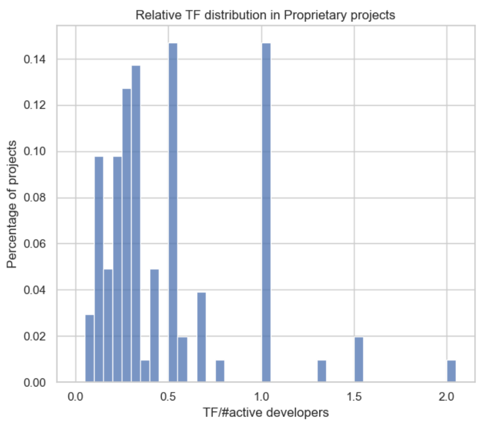
The relative TF were obtained by dividing the TF value by the number of active developers in each project, thereby a relative TF closer to 1, or greater, is desirable. The distributions of the relative TF for each dataset are illustrated in Figures 6.10a and 6.10b. Notably, both datasets include entries with a relative TF exceeding 1. This indicates that these projects have successfully established a high degree of knowledge distribution. For projects that have not surpassed a relative TF of one, the majority fall within the range of 0.0 to 0.5.

To comprehend the correlation between the absolute and relative Truck Factors in comparison with the number of active developers in the two datasets, the data were visualized by two box-and-whisker plots generated by the Python Seaborn library. These plots are visualized in Figures 6.11 and 6.12. The data were also grouped by context and interval of active developers. This data is represented in Tables 6.3 and 6.4.

Our findings suggest that there is not a substantial difference between the two datasets, as seen by the box-and-whisker plots and the tables. However, a consistent trend is apparent in both datasets; as the number of active developers increases, the relative TF generally decreases.



(a) CS OSS



(b) CS prop

**Figure 6.10:** The frequency of relative TF values in Proprietary and Open-source projects.

**Table 6.3:** The relative TF for the proprietary and open-source projects. Categorized in intervals by the number of active developers.

| | Average relative TF (StD) | | | | |
|---|---|---|---|---|---|
| #Active developers | 1 | 2-5 | 6-10 | 11-20 | 21-50 |
| Open-source | 1.16 (0.64) | 0.55 (0.32) | 0.33 (0.18) | 0.15 (0.09) | 0.13 (0.11) |
| Proprietary | 1.27 (0.65) | 0.67 (0.41) | 0.33 (0.15) | 0.26 (0.13) | 0.17 (0.08) |

**Table 6.4:** The TF for the proprietary and open-source projects. Categorized in intervals by number of active developers.

| | Average TF (StD) | | | | |
|---|---|---|---|---|---|
| #Active developers | 1 | 2-5 | 6-10 | 11-20 | 21-50 |
| Open-source | 1.16 (0.64) | 1.43 (0.84) | 2.34 (1.35) | 2.06 (1.14) | 3.86 (3.40) |
| Proprietary | 1.27 (0.65) | 1.93 (1.00) | 2.04 (1.00) | 3.42 (1.80) | 4.67 (2.25) |

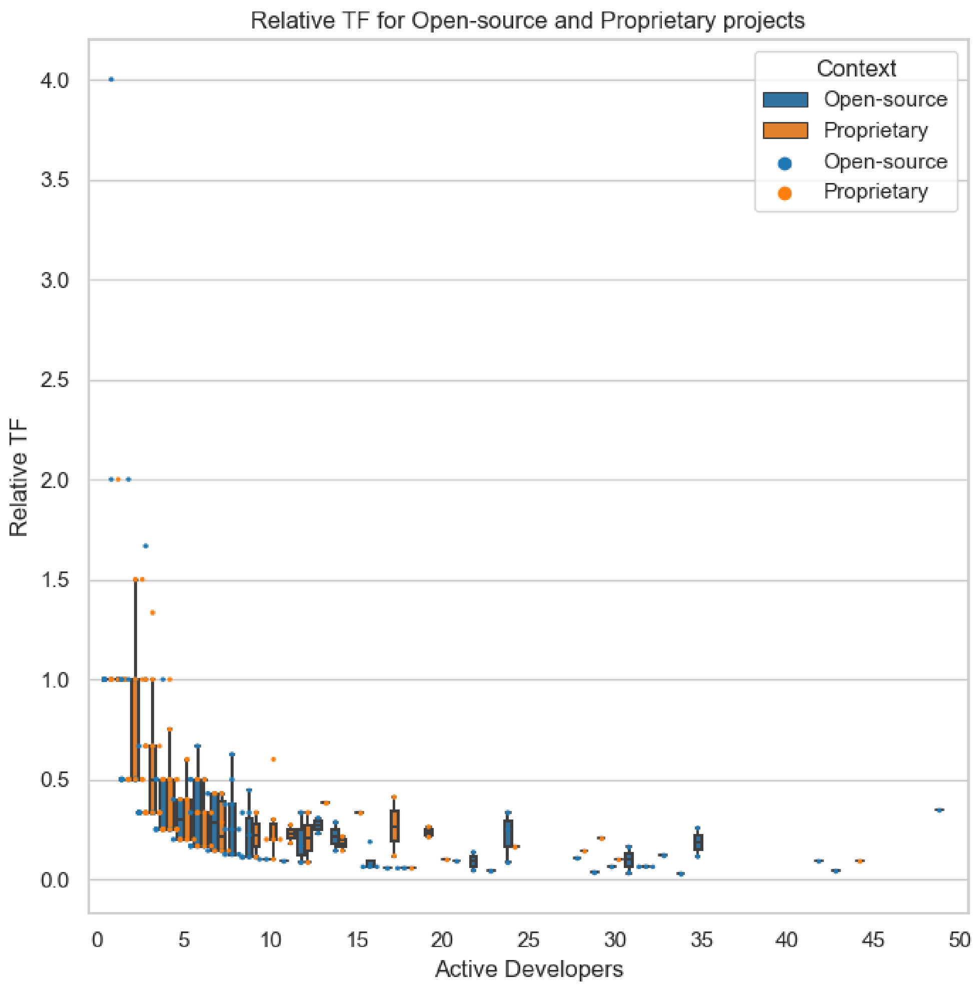


**Figure 6.11:** Comparison between relative TF values in proprietary and open-source systems.

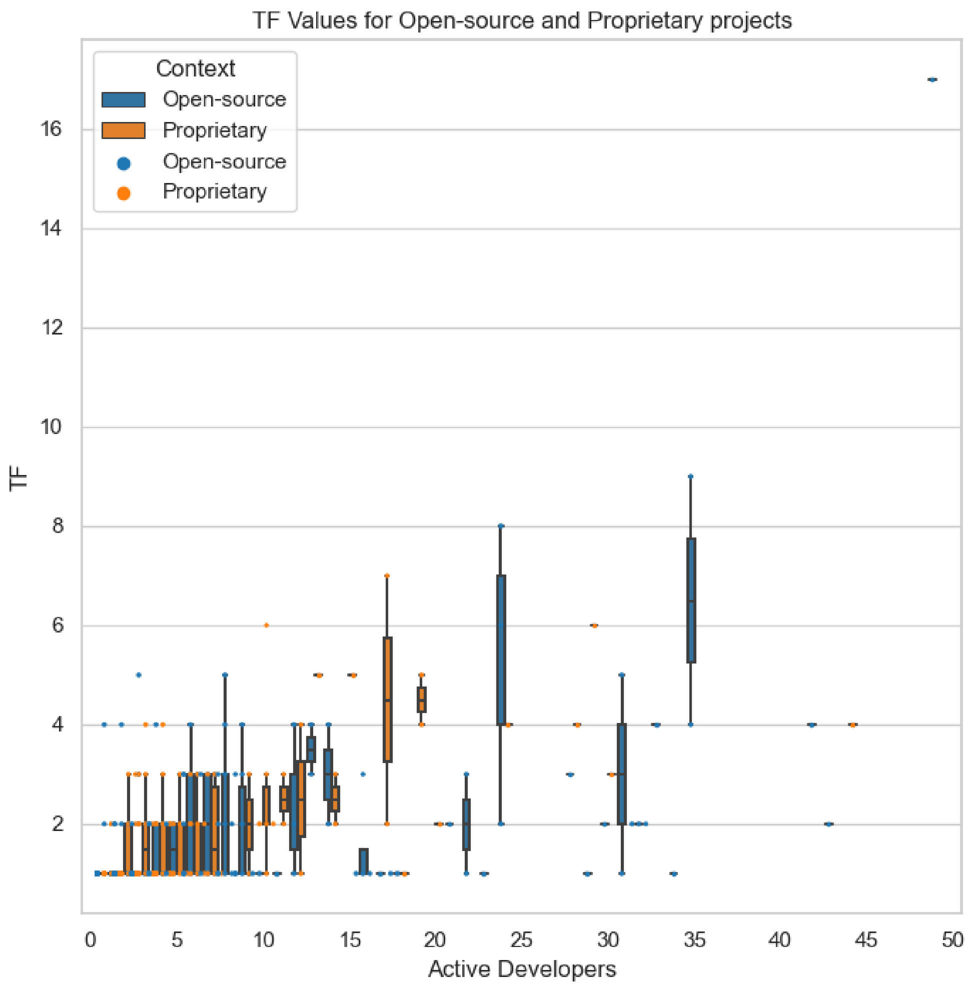


**Figure 6.12:** Comparison between TF values in proprietary and open-source systems.

# Chapter 7

# Discussion

This chapter addresses the research questions, starting with a discussion on RQ1, followed by an exploration of RQ2. Finally, potential threats to validity are considered.

## 7.1 RQ1: Comparing TF algorithms

Among the various algorithms considered, ALGO1 achieved the best results in the Truck Factor benchmark, surpassing all values obtained by the state-of-the-art algorithm proposed by Avelino *et al.* [4]. Consequently, the algorithm was chosen for phase 3. The selected threshold value of 30% is deemed appropriate, i.e., a developer must have contributed at least 30% of the LoC to qualify as an author of a file. The score ensures that the developer has been previously active with the file — beyond minor bug fixes or renaming activities, indicating a profound knowledge about the file and thereby fulfilling the criteria for being able to maintain and further develop the file. In combination with setting the other threshold value to 50%, i.e., the system is at risk when 50% of the files that previously had at least one author are orphaned, ALGO1 outperforms AVL on the TF oracle.

ALGO2, capturing the gist of the AVL v.1.2 by incorporating the functionality of neglecting developers who is author to a fraction of a project's files, yielded poor results compared to AVL v.1.2. This discovery was unexpected, since the performance between AVL v.1.0 and v.1.2 had clear difference when running on the AVL OSS dataset. Avelino introduced the feature in v.1.2 to exclude developers in open-source projects that had authorship distributions with long tails. This thesis did not investigate whether the TF Oracle dataset shares this characteristic. But, the results from ALGO2 suggest that this feature did not provide a better performance.

In comparison to all the algorithms assessed, ALGOCS yielded the least accurate results. However, the examination of threshold analysis, as detailed in Subsection 6.2, surprisingly revealed that relatively naive Truck Factor algorithms can provide reasonably accurate estimations. Notably, ALGO4 emerged as particularly effective, accurately estimating the Truck

Factor correctly for 65% of the projects.

## 7.2 RQ2: Comparing TF values in OSS and proprietary contexts

Previous research presented in Section 1.3 has utilized OSS projects to analyze Truck Factors for various repositories, likely due to their accessibility. This thesis extends previous research by also evaluating the TF distribution in proprietary projects. Section 6.3 demonstrated that there is no substantial difference between the relative and absolute TF in proprietary and OSS contexts.

In the introduction at Section 1.5, the topic of corporate increased influence in OSS communities was discussed. Our findings could signify the shift in OSS communities from being developed primarily by enthusiasts and amateurs to a scenario where employed developers at corporations play a more prominent role [14, 27]. Nonetheless, these findings support that OSS and proprietary projects share similar distributions of TF.

The presented data also indicates that larger organizations tend to have a lower relative TF. Organizations with 21-50 active developers exhibit an average relative Truck Factor of 0.13 for open-source projects and 0.17 for proprietary projects. The numbers suggest that the knowledge and expertise required to maintain and further develop the project are concentrated to a relatively few individuals.

The results presented in Subsection 6.2.1 align with the conclusions drawn by Borg *et al.* [7], emphasizing that industry practices lead to mainly dominant and marginal owners on a file-level in proprietary systems. Our findings extend this observation by highlighting that these practices are not exclusive to proprietary projects but are also prevalent in open-source projects. Our added observation with the relative TF suggests that collaborative knowledge sharing is not just limited to a file-level, but a project as a whole. This implies that the survival of software projects hinges on relatively few developers.

Avelino *et al.*[3] concluded that 19% of the 1,932 researched open-source projects have experienced being hit by a truck, with 41% of the affected projects managing to survive. The survivals were attributed to their communities' ability to attract new core contributors. In a proprietary context, where the aspiration is for all developers to be core contributors, this is not a desirable approach. Hence, in proprietary projects, more focus on knowledge sharing could entail a lower risk of being hit by the truck.

## 7.3 Threats to validity

In this chapter, we review the validity of our findings, focusing on four dimensions: Internal validity, construct validity, external validity, and reliability [26]. The aspect of internal validity explores factors that could have influenced our results. Construct validity represents the degree to which the operational measures used in research align with the intended objectives, i.e., how well our method answers the research questions. External validity assesses the generalizability of our findings and reliability examines the extent to which our results depend on us as researchers.

### 7.3.1 Internal validity

The thesis has used a newer version of the AVL algorithm than the one proposed in Avelino *et al.*'s paper [4] to establish a baseline. To ensure that the newer version works similarly to the one utilized in the original paper, contact was initiated with Avelino. However, which version of the AVL algorithm Ferreira *et al.* [12] utilized when AVL was concluded as state-of-the-art has not been identified. We have tried to get in touch with Ferreira, but to no avail. Therefore, it is a risk that our algorithms were benchmarked against another version of the algorithm than the one used in the paper.

During the tuning process, only the v1.2 version of the AVL algorithm with default settings was utilized. Tuning this algorithm similarly to our newly proposed algorithms would maybe not yield results where ALGO1 and ALGO3 outperforms AVL. The main reason why this has not been done is time limitations.

To establish the most precise algorithm, metrics such as mean absolute error and mean square error were used. The usage of these metrics was to identify the algorithm that provided results closest to the TF Oracle. However, these metrics did not measure the weight of the TF differences. In practice, the difference between estimating a TF of 22 compared to 24 is not as critical as a difference between TF 1 and 3. Considering an additional weighted error metric could have changed the outcome of Phase 2, and resulted in another algorithm and/or different configuration settings.

This thesis heavily relies on the TF Oracle introduced in Ferreira *et al.*'s work [12]. Efforts have been made to replicate the artifacts used in the paper. However, the absence of information regarding the specific commits to which the repositories were checked out makes it likely that this paper may not have used the exact same artifacts. Despite attempts to contact Ferreira for clarification, no successful communication was established.

In summary, all of these factors could have an impact on which algorithm we decided to use in Phase 3 and thus both affecting RQ1 and RQ2.

### 7.3.2 Construct validity

This thesis main construct is the TF. It refers to the number of key team members who, if they were to be hit by a proverbial truck (i.e., leave the project suddenly due to unforeseen circumstances), would significantly impact the project's progress and success. This constructs description is vague where significant impact is not defined. While the term 'significant impact' lacks a precise definition in the construct, this paper sheds light on a noteworthy observation: project files are commonly developed by a single developer. The potential loss of key developers can noticeably impede progress, particularly when considering that, on average, 58% of a developer's time is dedicated to understanding existing code [30]. Taking into account the additional time required to integrate new developers into a file, module, or system, we assert that the truck factor TF does not undermine the validity of this thesis.

The TF oracle is the construct we use for true TF scores. However, its validity can be questioned. The TF Oracle relies on individuals' responses to a questionnaire and not real events. Therefore, the tuning phases presented in this thesis could have been tuned to an inaccurate state which does not represent when a project has been 'hit'.

The algorithm utilizes contributed LoC as the only metric to determine a developer's knowledge about a file. In reality, a developer might be familiar with a file even if they have not made any changes to it, simply by working in its vicinity. Consequently, a potential author in reality could be a developer who has not made any direct changes to a file. In contrast to Jabrayilzade *et al.* [19], our thesis has not taken this into consideration.

### 7.3.3 External validity

As of our knowledge, the dataset used as the Oracle is currently the best publicly available. However, the Oracle used for the tuning only included 35 repositories. Consequently, the reliability of the algorithm's thresholds may not be optimal. This is why the analysis of the file ownership distribution was conducted.

To establish the TF distributions in the different development contexts, ALGO1 was run on 102 proprietary and 195 open-source projects of different sizes and using different programming languages. These datasets were queried from CodeScene's customers and could thereby be biased. Given CodeScene's role as a provider of codebase analysis tools, their customers and thereby the collected data may not comprehensively capture the diversity of projects in the broader reality.

The distribution of active developers in both sets is skewed towards relatively small values, averaging 7.5 for CS Prop and 8.5 for CS OSS. This suggests that our results might not generalize to projects with a higher number of active developers. This could impact the validity of the TF trend, which we investigate in RQ2. However, in the interval [1-8] active developers, we believe that our findings support that open-source and proprietary projects tend to share similar characteristics of the distribution of TF.

This thesis has had the approach to establish one algorithm that could handle all projects, no matter the languages used, the size of the organization or in which context it was developed. The thesis has, therefore, not delved into the topic of whether the algorithm should treat these aspects differently. Our findings have shown that, in general, the distribution of Truck Factors is similar among proprietary and open-source projects. However, the paper has not explored the distribution based on other aspects.

We have not delved into the importance of specific files or modules. There could be modules in a system that are critical and require specific expertise, i.e. a relatively few files could entail that the Truck Factor should be less than the one provided by the algorithm. Therefore, we consider that our algorithm could be used to provide an organization with insights, but without claiming that the TF is the definitive truth.

### 7.3.4 Reliability

The source code, which we have utilized together with the data, has been open-sourced and is available on GitHub. Therefore, anyone can scrutinize our results — supporting the reliability of our work.

Initially, contact with Avelino was established, and we received configuration files for the v1.0 version of the AVL algorithm. Our GitHub repository does not contain those files. However, since the v1.2 version of the AVL algorithm was used instead, which did not use these configuration files, and the artifact's commit id is presented, the artifacts and the results should be possible to recreate.

# Chapter 8

# Conclusion

This chapter offers the conclusion to the research question posed in this thesis. Additionally, potential directions for further research are presented.

## 8.1 Conclusion

The primary goal of this thesis is to lay the groundwork for the integration of a new Truck Factor algorithm into CodeScene's product range. The proposed algorithm has then been run on 297 repositories in both open-source and proprietary projects to identify to which extent the distribution of Truck Factors in open source-projects generalize to proprietary systems.

This thesis has presented five novel algorithms for estimating TF for repositories, utilizing Lines of Code as heuristic. Of the algorithms, ALGO1 presented the most accurate results on the TF Oracle presented by Ferreira *et al.*[12] with the configurations; to be an author of a file set to 30% and remaining files without an author set to 50%. These thresholds were established through a file authorship analysis and by benchmarking it against the current state-of-the-art algorithm proposed by Avelino *et al.* [4].

ALGO1 was run on 195 open-source projects and 102 projects mined from CodeScene's data lake. The findings proposed that the distribution between the two different contexts are similar, and that the more active developers a project has, the lower the relative Truck Factor tends to be.

Our findings highlight that the knowledge distribution in both open-source and proprietary projects could be improved to establish an organization that is more robust to the sudden loss of developers.

## 8.2 Future Work

We believe that using additional metrics could provide an algorithm that better models the true knowledge within a software project. In the introduction of this thesis, papers published by Jaybrazilade *et al.* [19] and Haratian *et al.* [16] were presented, both of which contributes valuable insights.

Jaybrazilade *et al.* enhanced the AVL algorithm by incorporating additional metrics into the DOA equation, considering factors such as knowledge decay over time, the number of code reviews, and time spent in file-related meetings. Due to constraints such as data accessibility, time limitations, and the papers' lack of external validation, this specific approach was not investigated. Nonetheless, we believe that integrating these metrics could improve our proposed algorithm, offering a more accurate model of project knowledge and consequently yielding more realistic truck factor values.

The algorithms presented in this thesis have adopted Avelino *et al.*'s [4] naive approach, treating all files as equally important. However, real-world software projects often involve modules with varying importance based on business impacts. Additionally, the complexity of maintaining files can differ based on their dependencies. Files with numerous reverse dependencies are likely to have a higher level of complexity, making the system more vulnerable if the authors of these files were to abruptly leave the project. Haratian *et al.* [16] expanded the AVL algorithm by incorporating a weighting mechanism for files based on their dependency degrees. We propose further research to explore integrating this functionality into the algorithm presented in this thesis.

In the realm of Truck Factor algorithms, there is a consensus that an algorithm capable of deterministically establishing the Truck Factor for a repository would be valuable. However, pinpointing when a repository is actually "hit" poses a challenge, and there is a scarcity of empirical studies on this matter. The primary reference in this thesis, Ferreira *et al.* [12], heavily relies on surveys and consensus rather than real events. Therefore, we advocate for further research to be conducted in this area to enhance our understanding and develop algorithms grounded in real-world occurrences.

# Bibliography

[1] Ashley Aitken and Vishnu Ilango. A comparative analysis of traditional software engineering and agile software development. In *2013 46th Hawaii International Conference on System Sciences*, pages 4751–4760, 2013.

[2] S. W. Ambler. *IBM agility@ scale™: Become as agile as you can be.* IBM Global Services, New York, USA, 2010.

[3] Guilherme Avelino, Eleni Constantinou, Marco Tulio Valente, and Alexander Serebrenik. On the Abandonment and Survival of Open Source Projects: An Empirical Investigation. In *Proc. of the International Symposium on Empirical Software Engineering and Measurement*, pages 1–12, 2019.

[4] Guilherme Avelino, Leonardo Passos, Andre Hora, and Marco Tulio Valente. A Novel Approach for Estimating Truck Factors. In *Proc. of the 24th International Conference on Program Comprehension*, pages 1–10, 2016.

[5] Guilherme Avelino, Marco Valente, Andre Hora, and Leonardo Passos. Measuring and analyzing code authorship in 1 + 118 open source projects. *Science of Computer Programming*, 176, Mar. 2019.

[6] Barry Boehm. Making a difference in the software century. *Computer*, 41(3):32–38, 2008.

[7] Markus Borg, Adam Tornhill, and Enys Mones. U owns the code that changes and how marginal owners resolve issues slower in low-quality source code. In *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*, EASE '23, page 368–377, New York, NY, USA, 2023. Association for Computing Machinery.

[8] Preetha Chatterjee, Tushar Sharma, and Paul Ralph. Empirical standards for repository mining. In *Proceedings of the 19th International Conference on Mining Software Repositories*, MSR '22, page 142–143, New York, NY, USA, 2022. Association for Computing Machinery.

[9] D. Cooper, K.-J. Stol, and A. Oram. The innersource approach to innovation and software development. In *Adopting InnerSource*, chapter 1. O'Reilly Media, Inc, Sebastopol, CA, USA, 2018.

[10] Valerio Cosentino, Javier Luis Cánovas Izquierdo, and Jordi Cabot. Assessing the bus factor of Git repositories. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 499–503, March 2015. ISSN: 1534-5351.

[11] Henry Edison, Noel Carroll, Lorraine Morgan, and Kieran Conboy. Inner source software development: Current thinking and an agenda for future research. *Journal of Systems and Software*, 163:110520, 2020.

[12] Mívian Ferreira, Thaís Mombach, Marco Tulio Valente, and Kecia Ferreira. Algorithms for estimating truck factors: a comparative study. *Software Quality Journal*, 27(4):1583–1617, December 2019.

[13] Thomas Fritz, Gail C. Murphy, Emerson Murphy-Hill, Jingwen Ou, and Emily Hill. Degree-of-knowledge: Modeling a developer's knowledge of code. *ACM Trans. Softw. Eng. Methodol.*, 23(2), Apr 2014.

[14] Jesus Gonzalez-Barahona and Gregorio Robles. Trends in free, libre, open source software communities: From volunteers to companies. *it - Information Technology*, 55, 10 2013.

[15] Jesus M. Gonzalez-Barahona and Gregorio Robles. Revisiting the reproducibility of empirical software engineering studies based on data retrieved from development repositories. *Information and Software Technology*, 164:107318, 2023.

[16] Vahid Haratian, Mikhail Evtikhiev, Pouria Derakhshanfar, Eray Tüzün, and Vladimir Kovalenko. Bfsig: Leveraging file significance in bus factor estimation. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1926–1936, 2023.

[17] Wilhelm Hasselbring. Benchmarking as empirical standard in software engineering research. In *Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering*, EASE '21, page 365–372, New York, NY, USA, 2021. Association for Computing Machinery.

[18] Lawrence Hecht and Libby Clark Clark. Survey: Open source programs are a best practice among large companies. https://thenewstack.io/survey-open-source-programs-are-a-best-practice-among-large-companies/, Jun 2021. Accessed on 2023-09-15.

[19] Elgun Jabrayilzade, Mikhail Evtikhiev, Eray Tüzün, and Vladimir Kovalenko. Bus factor in practice. In *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice*, ICSE-SEIP '22, page 97–106, New York, NY, USA, 2022. Association for Computing Machinery.

[20] Oualid Ktata and Ghislain Lévesque. Agile development: Issues and avenues requiring a substantial enhancement of the business perspective in large projects. In *Proceedings of the 2nd Canadian Conference on Computer Science and Software Engineering*, C3S2E '09, page 59–66, New York, NY, USA, 2009. Association for Computing Machinery.

[21] Greg Lewis and Joseph Soroñgon. Industries with the highest (and lowest) turnover rates. https://www.linkedin.com/business/talent/blog/talent-strategy/industries-with-the-highest-turnover-rates, Aug 2022. Accessed on 2023-11-15.

[22] Paul Petrone. See the industries with the highest turnover (and why it's so high). https://www.linkedin.com/business/learning/blog/learner-engagement/see-the-industries-with-the-highest-turnover-and-why-it-s-so-hi, Mar 2018.

[23] Filippo Ricca and Alessandro Marchetto. Are Heroes common in FLOSS projects? In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '10, pages 1–4, New York, NY, USA, September 2010. Association for Computing Machinery.

[24] Filippo Ricca, Alessandro Marchetto, and Marco Torchiano. On the Difficulty of Computing the Truck Factor. In Danilo Caivano, Markku Oivo, Maria Teresa Baldassarre, and Giuseppe Visaggio, editors, *Product-Focused Software Process Improvement*, Lecture Notes in Computer Science, pages 337–351, Berlin, Heidelberg, 2011. Springer.

[25] Peter C. Rigby, Yue Cai Zhu, Samuel M. Donadelli, and Audris Mockus. Quantifying and mitigating turnover-induced knowledge loss: case studies of chrome and a project at avaya. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 1006–1016, New York, NY, USA, 2016. Association for Computing Machinery.

[26] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, Apr 2009.

[27] Diomidis Spinellis and Vaggelis Giannikas. Organizational adoption of open source software. *Journal of Systems and Software*, 85:666–682, 03 2012.

[28] Marco Torchiano, Filippo Ricca, and Alessandro Marchetto. Is my project's truck factor low? theoretical and empirical considerations about the truck factor threshold. In *Proceedings of the 2nd International Workshop on Emerging Trends in Software Metrics*, WETSoM '11, pages 12–18, New York, NY, USA, 2011. Association for Computing Machinery.

[29] Jóakim v. Kistowski, Jeremy A. Arnold, Karl Huppler, Klaus-Dieter Lange, John L. Henning, and Paul Cao. How to build a benchmark. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, ICPE '15, page 333–336, New York, NY, USA, 2015. Association for Computing Machinery.

[30] Xin Xia, Lingfeng Bao, David Lo, Zhenchang Xing, Ahmed E. Hassan, and Shanping Li. Measuring Program Comprehension: A Large-Scale Field Study with Professionals. *IEEE Transactions on Software Engineering*, 44(10):951–976, 2018.

[31] K. Yamashita, Y. Kamei, N. Ubayashi, S. McIntosh, and A.E. Hassan. Revisiting the applicability of the pareto principle to core development teams in open source software projects. In *International Workshop on Principles of Software Evolution (IWPSE)*, volume 30-Aug-2015, pages 46–55 – 55, (1)Kyushu University, 2015.

[32] Nico Zazworka, Kai Stapel, Eric Knauss, Forrest Shull, Victor R. Basili, and Kurt Schneider. Are Developers Complying with the Process: An XP Study. In *Proceedings of the*

*2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '10, pages 1–10, New York, NY, USA, September 2010. Association for Computing Machinery.

# Appendices

# Appendix A
# Detailed setup

## A.1 TF Oracle

| Repository | GitHub Link | Commit-id | Date of commit |
|---|---|---|---|
| Alexreisner/geocoder | https://github.com/Alexreisner/geocoder.git | a6cc40b094e2f07577a10d870efbba8b35c9b62f | 2015-08-14 |
| androidannotations/androidannotations | https://github.com/androidannotations/androidannotations.git | b1e1903307d143b22f985d61c8b8e552d9c1c291 | 2016-09-09 |
| Atom/atom-shell | https://github.com/Atom/atom-shell.git | e296f6daa27fc54a8204ebb3ecd5d104d3e74696 | 2015-08-14 |
| Bjorn/tiled | https://github.com/Bjorn/tiled.git | f7e77d2c554ecd79153fbb4a03ef6dd05364be51 | 2015-08-13 |
| Cantino/huginn | https://github.com/Cantino/huginn.git | d40a372c8357e1dd57ab5de4c95c7e87a5d44f5e | 2016-09-14 |
| Capistrano/capistrano | https://github.com/Capistrano/capistrano.git | bf45cbafe5ef396933c81f334bdbc8c6d32a38c2 | 2016-09-09 |
| capnproto/capnproto | https://github.com/capnproto/capnproto.git | 21e7b91ecbfda217265bcbb1ef92b7097f6813f5 | 2015-08-14 |
| Celluloid/celluloid | https://github.com/Celluloid/celluloid.git | 35374b2a0cd20da9c332cd74cab8d1de941ecf6b | 2015-08-13 |
| chef/chef | https://github.com/chef/chef.git | bacb2ff93ccc2e14a0b721988e241a1d07f70795 | 2015-08-13 |
| D3/d3 | https://github.com/D3/d3.git | 55a8388d728315fd50f11ca5ab944b01296e4681 | 2016-09-12 |
| Deis/deis | https://github.com/Deis/deis.git | e5f349dd9fb090aa993ad74758b3bbf038d44d23 | 2016-09-14 |
| Dropwizard/metrics | https://github.com/Dropwizard/metrics.git | 3da5c88729eb4d08c914c6dd231a4eb7b97e0936 | 2015-08-03 |
| Facebook/osquery | https://github.com/Facebook/osquery.git | 43cf5f1a0ae7731e2780e45e078dd0c04d10f0bf | 2015-08-14 |
| Gruntjs/grunt | https://github.com/Gruntjs/grunt.git | 2ddec56192ba6599cd3aeb0dc812f54cfe83c07b | 2015-05-28 |
| Ipython/ipython | https://github.com/Ipython/ipython.git | f2d072e9f357dcee0001f1aa2fe88c6494e5dd1f | 2015-08-12 |
| Junit-team/junit4 | https://github.com/Junit-team/junit4.git | 41d44734f41aba0cf6ba5a11ff5d32ffed155027 | 2016-07-18 |
| Kennethreitz/requests | https://github.com/Kennethreitz/requests.git | 5524472cc76ea00d64181505f1fbb7f93f11cc2b | 2016-09-14 |
| Leaflet/leaflet | https://github.com/Leaflet/leaflet.git | 96d33b3a15c8168b2a2c850d334275bcc002dfa9 | 2015-08-11 |
| Less/less.js | https://github.com/Less/less.js.git | 19dc9e78b9abebfcf7ac13561f205b68094f9012 | 2015-07-25 |
| Mailpile/Mailpile | https://github.com/Mailpile/Mailpile.git | 2eebd5b604e737698ab64e2867e65684249120bb | 2015-08-12 |
| Netty/netty | https://github.com/Netty/netty.git | 75af257a62ab328edeeeed59b636d85910ba934c | 2015-08-15 |
| nicolasgramlich/AndEngine | https://github.com/nicolasgramlich/AndEngine.git | 85387a522c0494595ab79f5430b5bf5f5a8575b3 | 2013-12-11 |
| Pallets/flask | https://github.com/Pallets/flask.git | 270355abdcaa8f20ad6e7dd39f69279859a7055c | 2016-09-14 |
| Powerline/powerline | https://github.com/Powerline/powerline.git | 11c1e07b1220453120602aafd250ae2cfcbdf007 | 2015-08-08 |
| Puphpet/puphpet | https://github.com/Puphpet/puphpet.git | 060192945a3576f8ae511466b8d6a4c9a344aa3d | 2015-08-11 |
| ReactiveX/RxJava | https://github.com/ReactiveX/RxJava.git | adfabec8fb740fc873ece843736246742fdaba7c | 2015-08-12 |
| Requirejs/requirejs | https://github.com/Requirejs/requirejs.git | 51d005fd962d57d5434a579f81f22bf51cdfb0a0 | 2016-09-05 |
| Respect/Validation | https://github.com/Respect/Validation.git | 007e37b57067341c33cb9697887af21f833f9189 | 2015-01-30 |
| Ruby-grape/grape | https://github.com/Ruby-grape/grape.git | 220c345dff9602e431ac780abcb98dbb24293395 | 2016-09-14 |
| Saltstack/salt | https://github.com/Saltstack/salt.git | c34c6b992feccb6b1a26fab1b5c137eb7192fb3d | 2015-08-15 |
| Sass/sass | https://github.com/sass/sass.git | dcf09265a0cc5006bba82d1c3e4d46d7134d7896 | 2016-03-04 |
| SFTtech/openage | https://github.com/SFTtech/openage.git | f4199f4738e6e039fff16837c7afc44519d65736 | 2015-08-14 |
| Symfony/symfony | https://github.com/Symfony/symfony.git | 017e88b6a15a679e9c811d5ebcd47ab68efc60e5 | 2016-09-15 |
| Thoughtbot/paperclip | https://github.com/Thoughtbot/paperclip.git | 47b540d5bc3f1da1f789d2bf421adaeb3cf6fcd6 | 2015-08-03 |
| Tornadoweb/tornado | https://github.com/Tornadoweb/tornado.git | 6e96e7781bd0073de5964f49a98241b81a7aa15a | 2015-08-10 |

**Table A.1:** Commits used for Ferreira's TF oracle [12].

## A.2 AVL Dataset

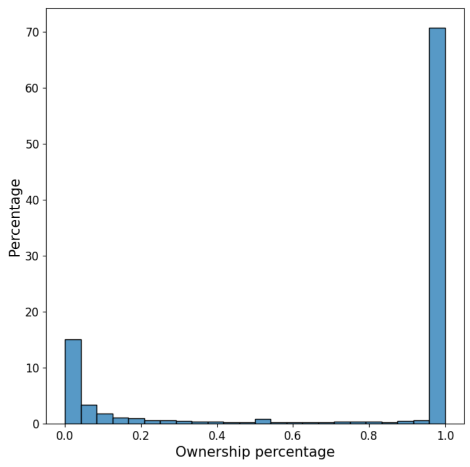
| fzaninotto/Faker | https://github.com/fzaninotto/Faker.git | 9625956ef3946a36b75fdc91e41bb12568aa2f39 | 2015-02-23 |
|---|---|---|---|
| Leaflet/Leaflet | https://github.com/Leaflet/Leaflet.git | a8f8f28911e5fa85d1afc677e2562066d5600b28 | 2015-02-21 |
| mailpile/Mailpile | https://github.com/mailpile/Mailpile.git | 97143a2971b00cf6340f19371d13b11fbdc468e4 | 2015-02-24 |
| FriendsOfPHP/PHP-CS-Fixer | https://github.com/FriendsOfPHP/PHP-CS-Fixer.git | f0cb37452e8c8a4bc38d2c5a228b393f8b72ab33 | 2015-02-25 |
| ratchetphp/Ratchet | https://github.com/ratchetphp/Ratchet.git | 21fbef177389ea7d573e774cb0b854701ea09db0 | 2015-01-25 |
| ReactiveX/RxJava | https://github.com/ReactiveX/RxJava.git | 4778d00d517c667b6b234189d70ff36e1e4ba924 | 2015-02-24 |
| silexphp/Silex | https://github.com/silexphp/Silex.git | d1cd7c132ee4f96466782ceec206313009c59bad | 2015-01-20 |
| Respect/Validation | https://github.com/Respect/Validation.git | fa67f0b0301082e670fa4bf4f4df94da47744f66 | 2015-02-20 |
| github/android | https://github.com/github/android.git | 59b0c285f165904ee4f944177ec89f92ee51929d | 2015-02-24 |
| excilys/androidannotations | https://github.com/excilys/androidannotations.git | 5ff1a1920e9b7b778b2cd116b070ad7b9ce835c2 | 2015-02-23 |
| atom/atom-shell | https://github.com/atom/atom-shell.git | 72d6a13e9e5f33d4811e558a1699d6ed04b93294 | 2015-02-19 |
| meskyanichi/backup | https://github.com/meskyanichi/backup.git | 98038e45cc9c849e4ccc385944d71ff8629abe3a | 2015-01-14 |
| sampsyo/beets | https://github.com/sampsyo/beets.git | ccbe9079718160a89a60a70fe6153028bba7b176 | 2015-02-24 |
| bitcoin/bitcoin | https://github.com/bitcoin/bitcoin.git | 34e5015cd21e27c1bf635d92531afac93f553096 | 2015-02-21 |
| sandstorm-io/capnproto | https://github.com/sandstorm-io/capnproto.git | 0e4ae5672fae3c8fc294340b1c4eaef13dc35174 | 2015-02-19 |
| celluloid/celluloid | https://github.com/celluloid/celluloid.git | 6a5c9cd5791306e6c09afbb9be1e8a2d5512204d | 2015-01-10 |
| chef/chef | https://github.com/chef/chef.git | fd849a49e89994d39478a792d1fb1932cc04fa22 | 2015-02-20 |
| clojure/clojure | https://github.com/clojure/clojure.git | 2e76c57d3b7672ce5eca096ed2059d9a1dd379d0 | 2015-01-14 |
| composer/composer | https://github.com/composer/composer.git | 475c84d4b1632653b115a4846125f276119db259 | 2015-02-25 |
| mbostock/d3 | https://github.com/mbostock/d3.git | 3abb00113662463e5c19eb87cd33f6d0ddc23bc0 | 2015-02-11 |
| dropwizard/dropwizard | https://github.com/dropwizard/dropwizard.git | 071b8104c0cf788264594899386ea5bfbf28d910 | 2015-02-25 |
| elasticsearch/elasticsearch | https://github.com/elasticsearch/elasticsearch.git | 82beae9c0c3ef181827a2b82b2d7145a50585cde | 2015-02-25 |
| emberjs/ember.js | https://github.com/emberjs/ember.js.git | 5d853cefaec16a58af72f380bbc927474eb88c63 | 2015-02-25 |
| strongloop/express | https://github.com/strongloop/express.git | 51f960f2977566f8d671fc0e8154466a1b3d78ca | 2015-02-23 |
| fog/fog | https://github.com/fog/fog.git | 46cea6f66d4db9015b3d98f9397e1cb4763c616a | 2015-02-25 |
| justinfrench/formtastic | https://github.com/justinfrench/formtastic.git | 0de7941dfd37c1391452320696be7516a8782d9c | 2015-01-08 |
| bumptech/glide | https://github.com/bumptech/glide.git | 75599b7af05eeca1393c8f4267adb78e88e7821e | 2023-10-10 |
| gruntjs/grunt | https://github.com/gruntjs/grunt.git | f290002eb3cede1004dbaf783d2b7b19f5d61bd4 | 2015-02-24 |
| caskroom/homebrew-cask | https://github.com/caskroom/homebrew-cask.git | 0151aeb0a96e7dbbda6d1fee71fd19239d01a370 | 2015-02-25 |
| Homebrew/homebrew | https://github.com/Homebrew/homebrew.git | 6f67f419ed627df55129eb620a806979d48905ae | 2015-02-25 |
| ipython/ipython | https://github.com/ipython/ipython.git | 54ea57400290d71935ec3699bbc3da92cbf791fc | 2015-02-23 |
| jadejs/jade | https://github.com/jadejs/jade.git | a15ae6bb5795a5d33879db2c94aa26fbd5682b6f | 2015-02-25 |
| xetorthio/jedis | https://github.com/xetorthio/jedis.git | 00eefa4767f908a1f6bade33bfa329acdcdf875f | 2015-02-22 |
| joomla/joomla-cms | https://github.com/joomla/joomla-cms.git | 69d86cf14bdf036ec6c761b66257546ef8cc0c5b | 2015-02-25 |
| jquery/jquery | https://github.com/jquery/jquery.git | 2380028ec4a6a77401b867a51de26a3cb8e8d311 | 2015-02-17 |
| kivy/kivy | https://github.com/kivy/kivy.git | 170fd047266315d642089ade4497dd446ecbcd43 | 2015-02-22 |
| less/less.js | https://github.com/less/less.js.git | 7eab7e4d4d4de87d990d3301db65380b295c2f37 | 2015-02-25 |
| libgdx/libgdx | https://github.com/libgdx/libgdx.git | 92e27b72e9af588e875e431a9cb3effb8fec5bc1 | 2015-02-23 |
| github/linguist | https://github.com/github/linguist.git | 739b512ceef87f1255a2acb20e850e80f6d5f602 | 2015-02-25 |
| elasticsearch/logstash | https://github.com/elasticsearch/logstash.git | 786d2d2c1aa3b8a4a2f6c703a54ce447fd6a2dc5 | 2015-02-19 |
| spotify/luigi | https://github.com/spotify/luigi.git | 541f40810d4f9f87e146cb0259854231e0a5aa7c | 2015-02-25 |
| dropwizard/metrics | https://github.com/dropwizard/metrics.git | b22b8da49f735e7c43902583519e28695db0a9b7 | 2015-02-12 |
| Seldaek/monolog | https://github.com/Seldaek/monolog.git | bba433ac380c47523fab495a1322368381df0996 | 2015-02-25 |
| janl/mustache.js | https://github.com/janl/mustache.js.git | d4ba5a19d4d04b139bbf7840fe342bb43930aee3 | 2015-02-18 |
| netty/netty | https://github.com/netty/netty.git | 237b393a8e7c7f1812ccf2c9510a205fe7772a8e | 2015-02-24 |
| openframeworks/openFrameworks | https://github.com/openframeworks/openFrameworks.git | 515b35e3d799be5af9eb06979819118ff78a9a13 | 2015-02-25 |
| SFTtech/openage | https://github.com/SFTtech/openage.git | a0fe241d6beda5ba65cd8a9f06d70f4b74d3a862 | 2015-02-18 |
| facebook/osquery | https://github.com/facebook/osquery.git | b9dbcb254584a52282c105a139025c2f2ac8dcd5 | 2015-02-25 |
| thoughtbot/paperclip | https://github.com/thoughtbot/paperclip.git | 97c78efdce65b07c6425e76000f34f1909a257cb | 2015-01-11 |
| sebastianbergmann/phpunit | https://github.com/sebastianbergmann/phpunit.git | 7b9e68dfc2fccb6b167b09a605152b0f5fd6d871 | 2015-02-24 |
| powerline/powerline | https://github.com/powerline/powerline.git | 3bf484de25cc77b0fc8f04e77f3993f4c1e939a8 | 2015-02-21 |
| puphpet/puphpet | https://github.com/puphpet/puphpet.git | a4da7b71c5c80f8a1bfff6b29f8f6ac8f2b27233 | 2018-03-24 |
| rails/rails | https://github.com/rails/rails.git | 71fc7892399bcb3ca24eff0a8f528e3bc8d7d82d | 2015-02-25 |
| jrburke/requirejs | https://github.com/jrburke/requirejs.git | 356dcd81f9982a01e6b27b4fe3a9b5494379e52a | 2015-02-19 |
| resque/resque | https://github.com/resque/resque.git | 04d40b00da45efb9194b9e4ab1fa485dde4f558e | 2015-01-17 |
| saltstack/salt | https://github.com/saltstack/salt.git | de441aba3503d904377a298696116567e70ce01e | 2015-02-25 |
| scikit-learn/scikit-learn | https://github.com/scikit-learn/scikit-learn.git | a92662672525618f7bf79f93044e57d4195828ac | 2015-02-25 |
| getsentry/sentry | https://github.com/getsentry/sentry.git | e188acb154ba1de130e8122d6c7025dc34f92157 | 2015-02-24 |
| bjorn/tiled | https://github.com/bjorn/tiled.git | fd52c607fad1b300c347236d0eb1a41a46a135ff | 2015-02-24 |
| JohnLangford/vowpal$_w$*abbit* | https://github.com/JohnLangford/vowpal$_w$*abbit.git* | 1c6cb8065a86684558455f409d0133c0ac686ef4 | 2015-02-18 |
| wp-cli/wp-cli | https://github.com/wp-cli/wp-cli.git | 0925413f215eb1ffecb80b278fb424ef8bf2f02b | 2015-02-20 |
| yiisoft/yii2 | https://github.com/yiisoft/yii2.git | 384607832f371acb20991c553e60dd29d7dfeb9e | 2015-02-25 |

**Table A.2:** Exact commits in the AVL OSS dataset [5].

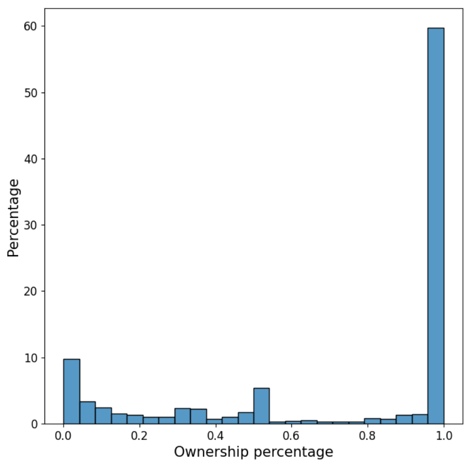
# Appendix B

# Detailed results

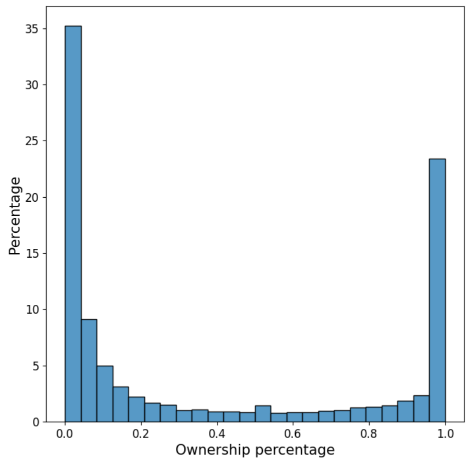
## B.1 File Authorships



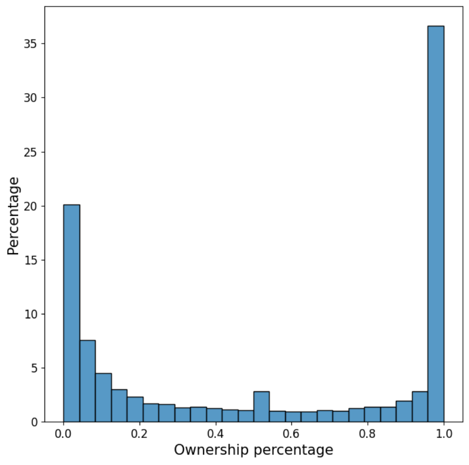
**(a)** Open-source: 1 active developer



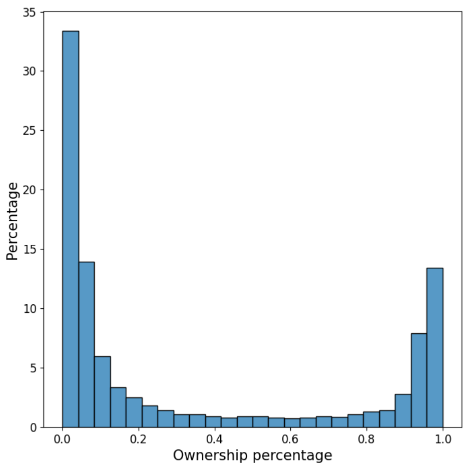
**(b)** Proprietary: 1 active developer



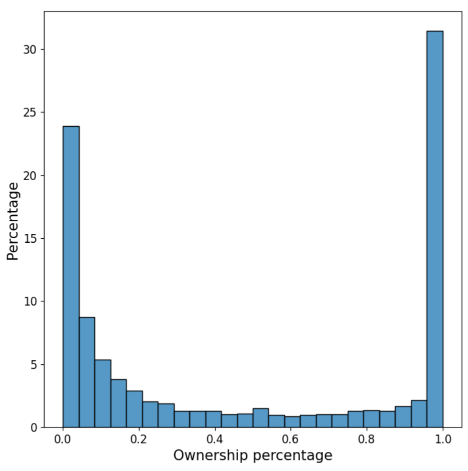
**(c)** Open-source: 2-4 active developers



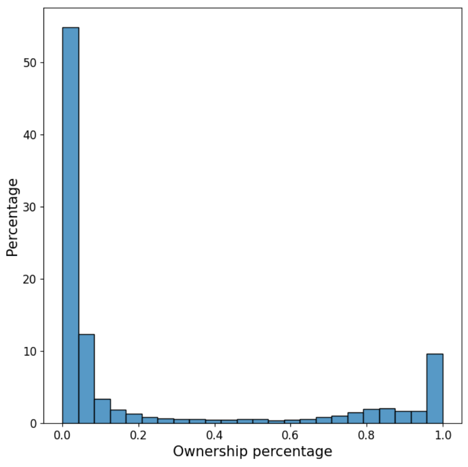
**(d)** Proprietary:2-4 active developer



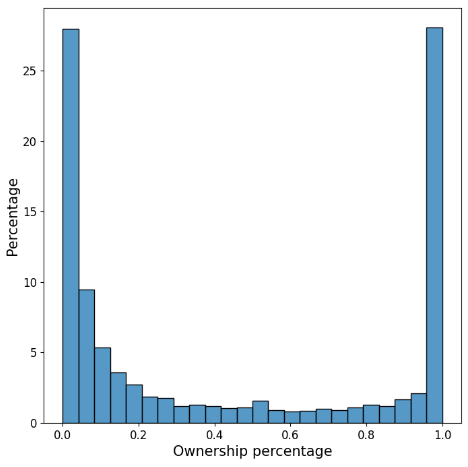
**(e)** Open-source: 5-9 active developers



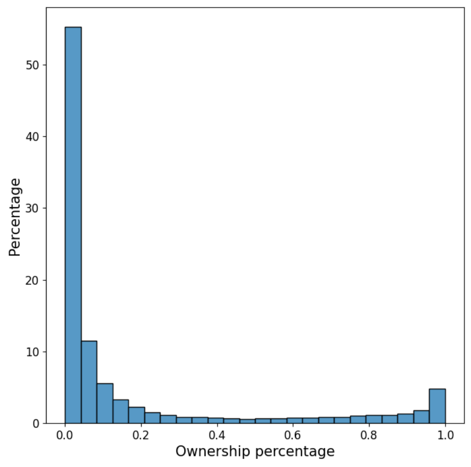
**(f)** Proprietary:5-9 active developer



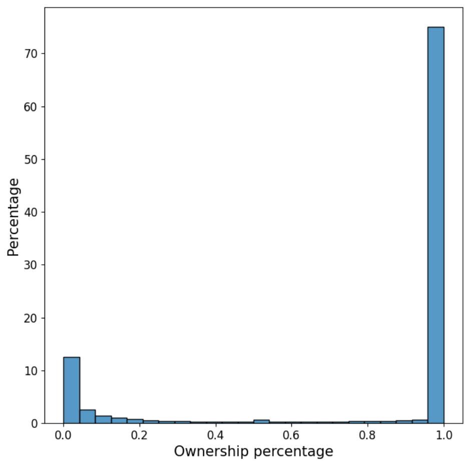
**(g)** Open-source: 10-24 active developers



**(h)** Proprietary:10-24 active developers



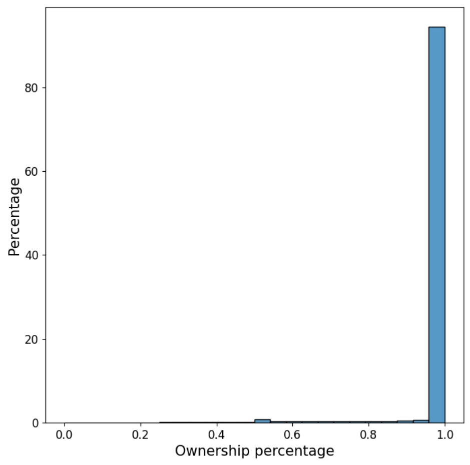
**(i)** Open-source: 25-50 active developers



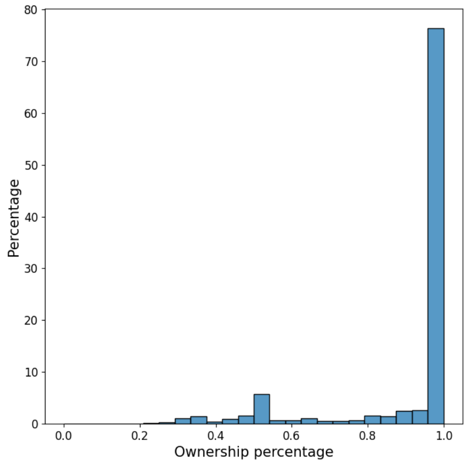
**(j)** Proprietary:25-50 active developers

**Figure B.1:** The distribution of the file authorship among the datasets for open-source and proprietary systems. The data is queried from CodeScene's customers' projects.

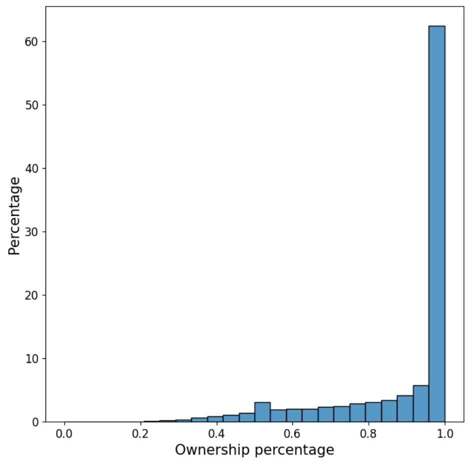
## B.2 Max File Authorships



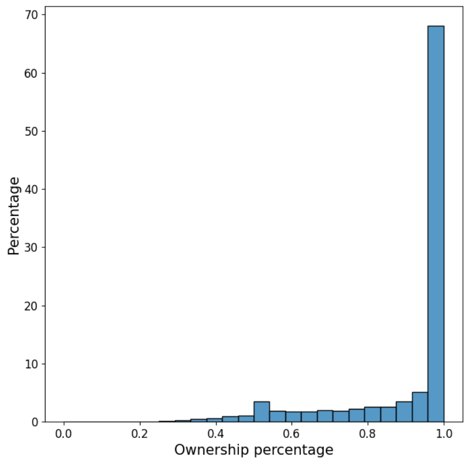
**(a)** Open-source: 1 active developer



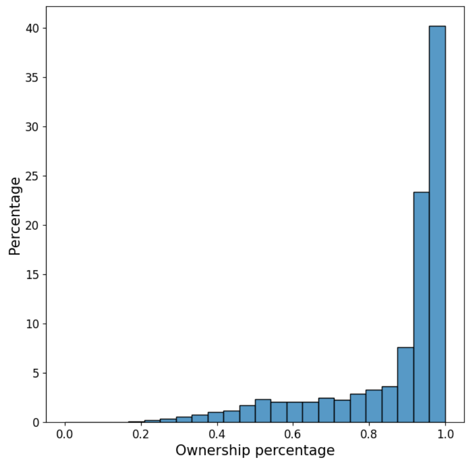
**(b)** Proprietary: 1 active developer



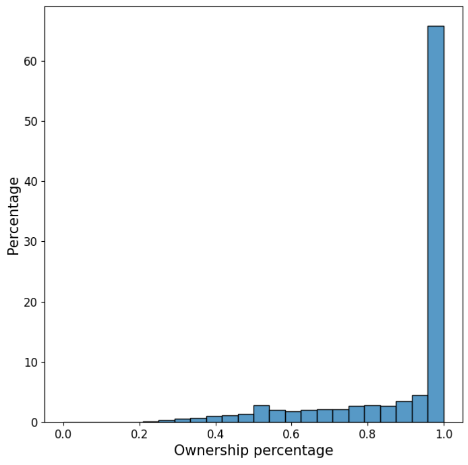
**(c)** Open-source: 2-4 active developers



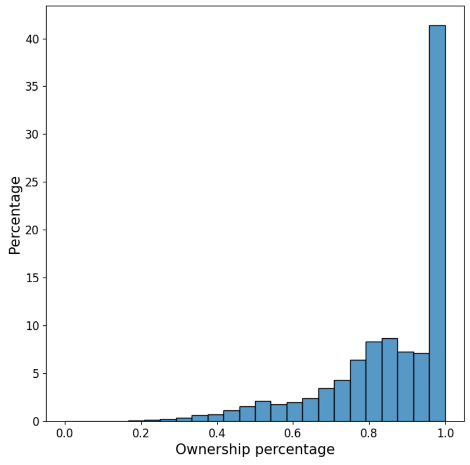
**(d)** Proprietary:2-4 active developer



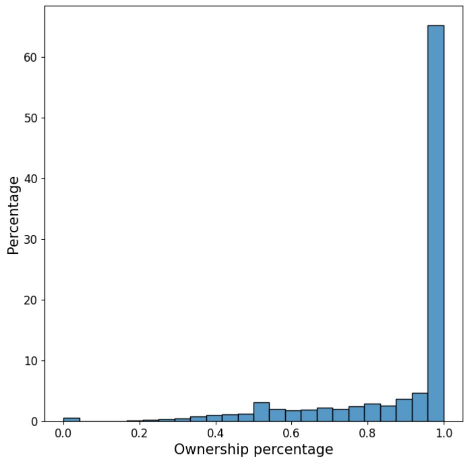
**(e)** Open-source: 5-9 active developers



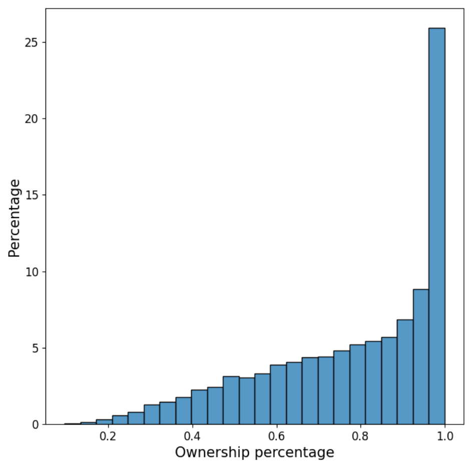
**(f)** Proprietary:5-9 active developer



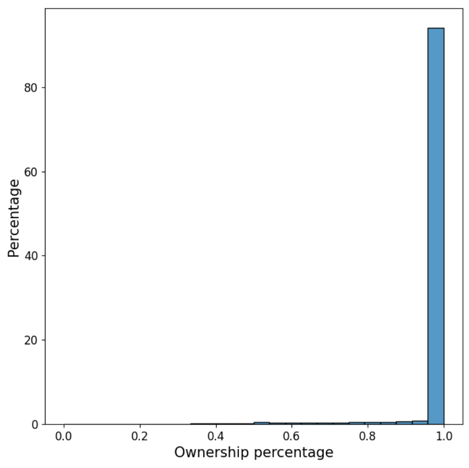
**(g)** Open-source: 10-24 active developers



**(h)** Proprietary:10-24 active developers



**(i)** Open-source: 25-50 active developers



**(j)** Proprietary:25-50 active developers

**Figure B.2:** The distribution of the maximum file authorship among the datasets for open-source and proprietary systems. The data is queried from CodeScene's customers' projects.

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2023-08-18

**EXAMENSARBETE** Driving Development Resilience:
Analyzing Truck Factors across Proprietary and Open-Source Projects
**STUDENT** Andreas Karlsson
**HANDLEDARE** Markus Borg (LTH)
**EXAMINATOR** Per Runeson (LTH)

# Att bli påkörd av en buss

POPULÄRVETENSKAPLIG SAMMANFATTNING **Andreas Karlsson**

Hur skulle din organisation reagera om någon oväntat blev påkörd av en buss? Utöver de personliga tragedierna väcker denna fråga tankar kring hur väl rustad organisationen är för att hantera en plötslig förlust av personal. Har organisationen förmågan att anpassa sig och fortsätta sin verksamhet?

Frågan om vad som händer med en organisation när någon blir påkörd av en buss kan upplevas överdriven. Men trots det så händer det att personal plötsligt lämnar sina roller och sina arbetsplatser. Detta innebär att organisationer behöver ha kunskaper och verktyg för att få en överblick och förståelse i hur beroende verksamheten är av specifika individer.

Inom mjukvaruutveckling så finns mycket av systemets kunskap inom varje utvecklare. Agila arbetsmetoder har premierat mindre dokumentation och mer informella kontaktkanaler vilket har medfört att organisationer har blivit mer beroende av sina utvecklare. Tidigare forskning från öppna källkodsprojekt har identifierat att många system är byggda av relativt få utvecklare. Detta medför att utvecklignen av systemen kan möta svåra problem om någon/några av dessa utvecklare plötsligt skulle lämna projektet.

Mitt exjobb handlar om att ta fram en algoritm som kan räkna ut Bussfaktorn i ett mjukvaruutvecklingsprojekt. Det vill säga en algoritm som kan identifiera det maximala antal nyckelpersoner som en organisation skulle klara av att plötsligt tappa utan att projektet inte längre skulle kunna gå att utveckla. Algoritmen är tänkt att användas för att hjälpa organisationer att identifiera vilken risk de har kopplat till plötsligt bortfall av utvecklare.

Den framtagna algoritmen användes på 195 öppna källkodsprojekt och 102 projekt hos företag för att undersöka om fördelningen av Bussfaktorerna var ungefär lika stora i de två olika typerna av projekt. Resultatet visade att de olika kontexternas fördelningar av Bussfaktorn var likartade. Det vill säga, att både öppna källkodsprojekt och proprietära projekt är likartade om vilken inneboende risk det finns i fall där utvecklare skulle lämna projekten.

Genom den framtagna algoritmen är tanken att företaget CodeScene ska implementera denna i sitt produktutbud. Detta skulle kunna hjälpa CodeScenes kunder att öka sin förståelse och möjlighet att agera på potentiella risker som de eventuellt inte var medvetna om tidigare, och på så sätt minska de negativa påföljderna om någon anställd abrupt skulle lämna organisationen. En konkret åtgärd kan vara att en organisation arbetar mer med att främja kunskapsfördelning för att uppnå ett ökat delat kodägarskap.