Thesis Dissertation

HOW STABLE IS TRUCK FACTOR ESTIMATION?

A SENSITIVITY ANALYSIS IN A SOFTWARE ECOSYSTEM

Andreas Sykopetritis

University of Cyprus

Computer Science Department

May 2026

UNIVERSITY OF CYPRUS

DEPARTMENT OF COMPUTER SCIENCE

HOW STABLE IS TRUCK FACTOR ESTIMATION?

A SENSITIVITY ANALYSIS IN A SOFTWARE ECOSYSTEM

Andreas Sykopetritis

Supervisor

Eleni Constantinou

The thesis was submitted for partial fulfillment of the requirements for obtaining the degree of bachelor's in computer science at the University of Cyprus

May 2026

Commented [EC1]: See the newer template and Mrs Georgiou's email on 5/5/26 mentioning that: Τονίζεται, ότι το Συμβούλιο του Τμήματός μας σε πρόσφατη Συνεδρία του, αποφάσισε όπως
οι φοιτητές/τριες στην τελική μορφή του κειμένου της Ατομικής Διπλωματικής Εργασίας θα
πρέπει ρητά να αναφέρουν την πιο κάτω δήλωση, ανάλογα με τη γλώσσα συγγραφής :
"*Δηλώνω ότι η παρούσα διατριβή και οποιοδήποτε συνοδευτικό λογισμικό αποτελούν*
δικό μου πρωτότυπο έργο, εκπονημένο σε πλήρη συμμόρφωση με τον Κώδικα
Δεοντολογίας για την Έρευνα του Πανεπιστημίου Κύπρου (https://www.ucy.ac.cy/legislation/kodikas-deontologias-kai-politikes/), και με πλήρη
ευθύνη εκ μέρους μου για την ακρίβεια, την ακεραιότητα και την εγκυρότητα όλου του
περιεχομένου.
I declare that this dissertation and any accompanying software are my own original
work, conducted in full compliance with the University of Cyprus Code of Conduct for
Research (https://www.ucy.ac.cy/legislation/kodikas-deontologias-kai-politikes/), and
with full accountability on my part for the accuracy, integrity, and validity of all
content."

“I declare that this dissertation and any accompanying software are my own original work, conducted in full compliance with the University of Cyprus Code of Conduct for Research (https://www.ucy.ac.cy/legislation/kodikas-deontologias-kai-politikes/), and with full accountability on my part for the accuracy, integrity, and validity of all Content.”

Acknowledgments

At this point, I would like to express my sincere thanks to my supervising professor, Ms. Eleni Constantinou, for her guidance, valuable observations, and continuous support throughout the preparation of this thesis. In addition, I would like to thank my family and friends for their encouragement, patience, and support, which contributed significantly to the completion of this effort.

Abstract

The Truck Factor (TF) is a widely used metric for estimating knowledge concentration risk in software projects, as it represents the minimum number of key contributors whose loss could seriously endanger a project's maintainability and continuity. Despite the popularity of the most widely used TF algorithm, its sensitivity to the parameters and thresholds it employs has not been sufficiently examined. In addition, TF estimation has rarely been studied in the context of software ecosystems where developers may be critical not only to individual projects but also to multiple interconnected repositories. This thesis investigates the stability of Truck Factor estimation under different methodological choices by applying Avelino's Truck Factor algorithm to the Cargo ecosystem. To achieve this, a dataset of Cargo packages (crates) was analyzed across multiple timelines, and the algorithm's results were compared under different parameter settings, including alternative threshold values and file-selection configurations. The goal was to examine the extent to which these methodological choices influence the resulting TF estimates and the identification of critical developers. This study is important because Truck Factor is often used as an indicator of project resilience and sustainability, yet its practical value depends on whether its estimates remain robust under reasonable variations in the estimation process. Moreover, in a dependency-heavy ecosystem such as Cargo, the departure of a critical contributor may affect not only one repository but also many downstream projects that rely on it. Therefore, understanding the stability of TF estimation is necessary both for interpreting the metric correctly and for assessing ecosystem-level maintenance risk. The results show that Truck Factor estimation is broadly stable at the aggregate level, but sensitive to parameter variation, with coverage thresholds producing the strongest changes and normalized Degree of Authorship thresholds producing moderate to substantial effects depending on the selected value.

Contents

Commented [EC2]: Developer knowledge???

Commented [EC3]: ? Check if this makes sense later on.

Section 1

Introduction

Commented [EC4]: @Antreas Sykopetritis review of Chapter 1 done. Very good!

1.1 Background and Motivation

Commented [EC5]: You need to back up your statements with references. e.g., who proved that in many software projects such knowledge is not equally shared among contributors? you need a reference for that.

Software systems are developed and maintained through the contributions of individuals who gradually gather important technical knowledge about the subject. In many software projects, however, such knowledge is not equally shared among contributors [2]; instead, it is often concentrated in a small number of developers who become critical to the project's evolution and long-term sustainability. This concentration of knowledge creates a significant risk. If key developers become inactive or leave the project, the project may face serious difficulties in maintenance, further development, and continuity. This problem is especially relevant in open-source software, where contributors often participate voluntarily, and their involvement may change over time. As a result, software engineering research has shown increasing interest in methods for identifying and measuring the dependence of projects on specific contributors.

One of the most widely known metrics for this purpose is the Truck Factor (TF). The Truck Factor represents the minimum number of developers whose sudden departure will put a project in serious danger. A low Truck Factor suggests that the project depends heavily on a few individuals, while a higher Truck Factor indicates that knowledge is more widely distributed. Because of its simplicity and practical interpretation, the Truck Factor has been used as an indicator of project resilience and knowledge concentration.

Several algorithms have been proposed to estimate the Truck Factor of a software project, most of them relying on the analysis of repository history to identify key contributors. In general, these approaches examine commits and file modification records to determine which developers are most strongly associated with particular files or components and then aggregate

this information to estimate the minimum number of contributors whose loss would seriously threaten the project. Although the algorithms follow this common idea, they differ in how they define file ownership, which files they consider relevant, and which thresholds they use to decide whether a developer is essential. As a result, Truck Factor estimation is usually an approximation based on a set of methodological choices.

Despite the algorithm's usefulness, Truck Factor estimation is not a straightforward process. Existing algorithms rely on assumptions about file ownership, the importance of different files within a project, and they do not pay attention to the timeline of the project. These methodological choices can influence the final Truck Factor value. However, the extent to which such choices affect the stability of the estimation has not been sufficiently studied.

Commented [EC6]: which algorithm's?

Commented [EC7]: Before discussing the existing algorithms' shortcomings, you should first give an overview of these algorithms in a paragraph right before this one. (not too many details here though)

In addition, most prior work examines Truck Factor at the level of individual projects. In practice, however, modern software is often developed within broader software ecosystems, where contributors may participate in multiple related projects. In such environments, a developer may carry critical knowledge across several projects at the same time. This means that the departure of a single contributor may have consequences for more than one repository. Therefore, studying Truck Factor in a software ecosystem can provide a better understanding of knowledge concentration risk.

The Cargo ecosystem, which supports the Rust programming language through its package manager and collection of crates, provides a suitable case study for this research. Although other software ecosystems also consist of many interconnected projects and active contributors, Cargo was selected because it is large enough to support meaningful analysis of knowledge concentration at both project and ecosystem level, while still being manageable in scope for a thesis study. In addition, its organization into crates offers a clear ecosystem structure that facilitates the identification of related projects and shared contributors. For these reasons, Cargo provides an appropriate setting for examining both project-level Truck Factor estimation and the broader distribution of key developers across an ecosystem.

Commented [EC8]: to be fair, many other ecosystems have these attributes. so why Cargo?

1.2 Research Problem and Objective

Although the Truck Factor is widely used to estimate knowledge concentration risk in software projects, its reliability depends on the assumptions and parameters of the estimation method.

Existing approaches, including Avelino's Truck Factor algorithm [1], assign ownership and identify key developers based on repository history and predefined thresholds. However, different parameter values or different selections of project files may lead to different Truck Factor results. It therefore becomes important to study if the estimated Truck Factor changes significantly when the algorithm settings change since this could indicate that the metric may be sensitive to the parameters used.

The main objective of this thesis is to study Truck Factor estimation at the software ecosystem level by applying and adapting Avelino's Truck Factor algorithm to the Cargo ecosystem. In particular, the thesis examines the sensitivity of the algorithm to the parameters and thresholds used in the estimation process, investigates the presence of Truck Factor developers in individual projects and their significance within the ecosystem as a whole, and analyzes the impact of their abandonment on dependent projects. Furthermore, in cases where projects survive the loss of their original Truck Factor developers, the thesis explores the origin of the new Truck Factor developers who subsequently undertake the maintenance of those projects.

Commented [EC9]: reference to the paper introducing the algorithm

Commented [EC10]: Is this the only scope of the thesis? To study the algorithm's sensitivity? Also apply it in an ecosystem setting?

Commented [EC11]: This we can review last when everything is written in your text.

1.3 Structure of the Thesis

The rest of this thesis begins in Section 2 by presenting the theoretical background of the study. We discuss software knowledge distribution, introduce the concept of Truck Factor, review existing Truck Factor estimation algorithms, and outline the main challenges and limitations associated with their usage.

In Section 3 we describe the research methodology that we followed, and explain the research design, the data collection process, the selection of projects from the Cargo ecosystem, the application of Avelino's algorithm, and the approach used for stability and sensitivity assessment.

In Section 4, we present the results from the analysis, and provide an overview of the selected projects, show the Truck Factor estimation results, examine the distribution of Truck Factor developers across the Cargo ecosystem, and analyze the sensitivity of the estimates to different parameters.

Within Section 5 we discuss the findings of the research, their implications, examine their significance, and present the threats to validity.

Section 6 concludes the thesis by summarizing the main findings, highlighting the contributions of the study, and suggests directions for future work.

Section 2

Background

2.1 Software Knowledge Distribution

Software development is a knowledge-intensive activity [3]. As projects evolve, contributors gradually accumulate understanding about the structure of the code, the rationale behind design decisions, the dependencies between modules, and the practices required to maintain the system correctly [3]. This knowledge is rarely distributed evenly across all contributors. Instead, some developers become more familiar with specific files, components, or architectural areas because they created them, modified them repeatedly, or coordinated changes around them [4],[5]. For this reason, software engineering research often uses concepts such as code ownership, authorship, and expertise as proxies for understanding how knowledge is distributed inside a project. Studies on code ownership have shown that ownership is an important characteristic of collaborative development and is related to the quality and evolution of software systems [4]. Likewise, research on source code familiarity has shown that knowledge is not captured only by the final state of the code but is also connected to the history of authorship and interaction around it [5].

In practice, knowledge concentration appears when a small number of developers are responsible for a large part of a system or for its most critical components. Such concentration may be efficient in the short term, because experienced contributors can make decisions quickly and maintain coherence in the codebase. However, when project knowledge is concentrated in a few individuals, the project becomes more vulnerable to turnover, inactivity, or reduced participation. If one of these key developers leaves, the remaining contributors may

Commented [EC12]: This is mentioned in [3]? if yes, add the citation at the end.

Commented [EC13]: One citation for this?

find it difficult to understand the affected parts of the code, review changes confidently, or continue maintenance with the same speed and quality. Therefore, knowledge distribution is directly related to project resilience and maintainability.

The issue becomes especially important in open-source software, where participation is usually voluntary, and contributor availability can change significantly over time [16]. In such projects, repository history often becomes the main observable source for inferring who knows what, because direct organizational information is limited or unavailable. As a result, many studies estimate expertise by mining version control systems and analyzing commits, file modifications, blame information, and related development activity [1]. These repository-based indicators are useful because they are available at scale, but they still remain approximations of actual knowledge. They reveal who changed the code, not necessarily who understands all its implications, who reviews it, or who makes important design decisions outside the repository.

Commented [EC14]: citation for this.

Commented [EC15]: Very good!!

Knowledge distribution must also be examined beyond the boundaries of a single repository. Modern software is often developed inside broader software ecosystems, where projects are interdependent and evolve together. Software ecosystem research describes such environments as collections of related software projects, components, and communities connected through technical and social dependencies [8]. In these settings, contributors may participate in several projects at once, and maintenance activities in one project may affect many others. Research on open-source software ecosystems has shown that collaboration often extends across project boundaries and that ecosystem-level maintenance may involve contributors from multiple communities [9]. Therefore, knowledge concentration in an ecosystem is not only a project-level issue, it may also reflect how critical expertise is distributed across multiple related projects.

Commented [EC16]: I would rephrase; knowledge concentration is an issue?

2.2 Truck Factor

One of the most widely known concepts used to capture knowledge of concentration risk is the Truck Factor. In simple terms, the Truck Factor of a software project is the minimum number of developers whose sudden departure would place the project in serious trouble. The idea offers an intuitive way to reason about project resilience: if a project has a Truck Factor of one, it depends heavily on a single person, if it has a higher Truck Factor, knowledge appears to be distributed among more contributors. Because of this intuitive interpretation, the metric has

become popular in both research and practice as an indicator of organizational and technical vulnerability.

The main appeal of the Truck Factor lies in its simplicity. It translates a complex socio-technical issue (knowledge concentration) into a single number that can be compared across projects or tracked over time. A low Truck Factor may signal the need for better documentation, knowledge sharing, mentoring, onboarding practices, or a broader distribution of maintenance responsibilities. A higher Truck Factor, by contrast, usually suggests that the project can better tolerate contributor loss. Nevertheless, the value should not be interpreted as an exact prediction of project failure. Rather, it is a risk indicator based on assumptions about what counts as relevant knowledge, how knowledge is represented, and when a project should be considered in danger. In addition, most Truck Factor estimation approaches simplify knowledge further by inferring it mainly from repository activity, especially commits and file modifications, even though important knowledge may also be developed through code reviews, discussions, and other forms of collaboration.

Commented [EC17]: Since you are discussing that a number is a simplification, it would be worth briefly mentioning another simplication of knowledge, aka assuming that knowledge comes only from commits.

The literature also shows that practitioners consider this problem meaningful. In a recent industrial study, engineers perceived the bus factor (TF) as an important issue in collaborative development, and the authors emphasized that identifying key engineers can help organizations manage the risks associated with sudden departure [6]. At the same time, recent theoretical work points out that different interpretations of the metric coexist in the literature. Some approaches emphasize redundancy, that is, how many people can leave before the project is harmed, while others emphasize criticality, that is, the smallest set of people whose loss would cause serious damage [7]. This distinction is important because two algorithms may use the same term while actually measuring slightly different things.

2.3 Truck Factor Estimation Algorithms

Several algorithms have been proposed to estimate the Truck Factor automatically from development data. The earliest line of work focused on identifying which developers are associated with which files and then simulating the effect of contributor loss on project coverage. The first algorithmic description is usually attributed to Zazworka et al. [10], later formalized by Ricca et al [11]. In this approach, the Truck Factor is derived by examining combinations of developers and checking whether their removal causes file coverage to fall below a predefined threshold. Although this work was important for establishing the problem,

it did not provide the strongest empirical support for real-world large-scale use since it's computationally demanding because it relies on combinations of contributors.

A later tool-oriented approach was proposed by Cosentino et al. [12]. Their method estimates developer knowledge for files and then aggregates these values to directory, branch, or project level. This approach is useful because it supports analysis at different granularity levels and allows simulation of what happens when one or more contributors disappear. Like other repository-mining approaches, however, it still depends on how knowledge is computed from version control data and on the thresholds used to decide whether a file or project is sufficiently covered.

Among existing approaches, the algorithm proposed by Avelino et al. [1] is one of the most influential and widely reused. It estimates file authorship using the Degree of Authorship (DOA) model, which combines three main factors: whether a developer originally created the file, how many changes that developer made to it, and how many changes were made by other developers. After computing and normalizing DOA values, the algorithm identifies the authors of each file using configurable thresholds. It then applies a greedy process: developers are ordered by the number of files they author, the top author is removed iteratively, and the Truck Factor is reached when more than 50% of the project files become abandoned, meaning that all their authors have been removed. To improve the quality of the estimate, Avelino et al. also perform preprocessing steps such as removing non-source files, excluding third-party code, and handling developer aliases. Their empirical study on 133 GitHub projects and the accompanying survey of project contributors gave important credibility to the approach and helped establish it as a state-of-the-art baseline.

Another important alternative is the algorithm by Rigby et al. [13]. Instead of commit-based authorship, it uses a blame-based approach to identify who last changed each line of code. In this method, a file becomes abandoned when a large proportion of its lines (90% in the formulation discussed by subsequent comparative work) are attributed to developers who have left. The algorithm simulates multiple disaster scenarios by randomly sampling groups of departing developers. This makes it conceptually different from Avelino et al.'s method, since it is non-deterministic and line-oriented rather than file-author oriented.

Comparative work by Ferreira et al. [14] examined major Truck Factor algorithms and found that the algorithm of Avelino et al. achieved the best accuracy among the approaches compared in their study. Even so, later research has continued to improve expertise identification and

Commented [EC18]: why not?

Commented [EC19]: Avelino et al.'s method

Commented [EC20]: Will not fix et al. from now on. When authors are >=3, we write the first author's surname and "et al.".

Truck Factor estimation. For example, recent work on source code expert identification showed that models incorporating additional variables, especially recency-related ones, can improve the identification of key developers and address weaknesses in the original DOA-based approach [15]. Other studies have also argued that repository-only approaches miss important knowledge channels such as code reviews, meetings, and other collaboration activities [5]. These developments suggest that Truck Factor estimation is an active research area rather than a settled problem.

2.4 Challenges and Limitations of Truck Factor Estimation

Although Truck Factor estimation is useful, it is not straightforward. A first challenge is that most algorithms rely heavily on version control system data as a proxy for knowledge. Repository history captures code creation and modification, but it does not fully capture architectural reasoning, informal mentoring, review activity, design discussions, or communication that happens through meetings, chat tools, and issue trackers. Fritz et al. [5] showed that developer knowledge is not explained only by authorship, while Jabrayilzade et al. [6] later argued that multimodal information can slightly improve estimation over VCS-only approaches. Therefore, a contributor may possess substantial project knowledge without appearing as a dominant author in repository history alone.

A second challenge concerns the sensitivity of the estimate to methodological choices. Truck Factor algorithms require decisions about which files to include, how to identify authorship, what thresholds to use, and when the project should be considered endangered. In Avelino's approach [1], for example, the result depends on DOA thresholds, on the exclusion of documentation and third-party code, and on the decision to declare the project endangered when author coverage falls below 50%. The authors themselves note threats related to lost version history, non-source files, third-party libraries, and developer aliases. In addition, feedback from surveyed developers suggested that the original method may overemphasize first authorship and may insufficiently reflect more recent maintainers. This is especially relevant in mature projects where original creators become less active, and maintenance responsibility shifts to newer contributors.

The third limitation is conceptual. Recent theoretical work argues that many existing approaches are based on coverage thresholds that are somewhat arbitrary and may fail to

capture fragmentation within a project. Under coverage-based reasoning, a project may appear relatively safe even when knowledge is split in a way that leaves critical integration points highly vulnerable. This line of work formalizes the underlying computational problem and shows that different formulations of the TF lead to different interpretations, each with its own limitations [7]. From this perspective, the Truck Factor is not a single universally defined quantity, but a family of related measures shaped by modeling assumptions.

Finally, most previous studies estimate Truck Factor at the level of individual repositories, even though much modern software development takes place inside ecosystems of interdependent projects. In software ecosystems, developers may contribute to several related repositories, and the loss of one contributor may affect multiple packages at once. Prior ecosystem research shows that projects in such environments evolve together, and that cross-project collaboration is common [9]. As a result, project-level Truck Factor may underestimate the broader risk created by developers whose knowledge spans many ecosystem components. This limitation provides strong motivation for studying Truck Factor not only within single projects, but also in the context of a software ecosystem such as Cargo.

Taken together, these challenges show that Truck Factor estimation is valuable but inherently approximate. The metric remains useful as an indicator of knowledge concentration risk, yet its meaning depends on the assumptions embedded in the estimation process. For this reason, it is important to examine not only the resulting Truck Factor values, but also how stable those values remain when algorithm parameters and thresholds decisions change.

Section 3

Research Methodology

3.1 Research Design

This thesis follows a quantitative empirical research design based on repository mining. The purpose of the study is to examine how the Truck Factor estimation changes when the parameters and thresholds of Avelino's algorithm [1] are varied, and to investigate how knowledge concentration appears not only at the level of individual projects but also across the Cargo software ecosystem. The study is therefore both project-oriented and ecosystem-oriented, first the Truck Factor is estimated for individual Cargo projects and second, the identified Truck Factor developers are analyzed in relation to their presence and importance across multiple projects.

The research was designed as a multi-stage analysis. In the first stage, a dataset of Cargo packages and their associated public repositories were collected and prepared for analysis. In the second stage, Avelino's Truck Factor algorithm was applied to each selected project in order to identify the developers whose departure would place the project at risk. In the third stage, the algorithm was executed repeatedly under different parameter settings to assess the stability and sensitivity of the estimated Truck Factor values. In the fourth stage, the identified Truck Factor developers were analyzed at ecosystem level to examine whether the same contributors played a critical role in more than one Cargo project and to evaluate the potential broader impact of their abandonment.

The unit of analysis in this study is the individual Cargo package repository, referred to as a project or crate, while the broader analytical context is the Cargo ecosystem. This dual perspective is important because the Truck Factor is traditionally estimated at the project level, yet modern open-source development often takes place in ecosystems where developers contribute across repository boundaries. Consequently, a developer identified as critical in one project may also hold important knowledge in several others, which means that the risk associated with that developer extends beyond a single repository.

The study is guided by the general objective of evaluating the stability of Truck Factor estimation and understanding the ecosystem-wide role of Truck Factor developers. More specifically, the methodology was designed to address the following concerns: whether different algorithmic choices produce substantially different Truck Factor values, whether the set of identified Truck Factor developers changes under different settings and how Truck Factor developers are distributed across Cargo projects. In this way, the methodology combines algorithmic replication, sensitivity analysis, and ecosystem-level contributor analysis.

Because the study relies on publicly available repository data, it does not attempt to directly measure actual developer knowledge through interviews or surveys. Instead, it adopts the common repository-mining assumption used in prior Truck Factor research. Repository history provides an observable proxy for knowledge ownership and distribution. This means that the findings should be interpreted as estimates of knowledge concentration based on development history rather than as direct measurements of all knowledge that may exist in a project.

3.2 Data Collection

Commented [EC21]: At each step in this section you need to mention the input number of projects, and after the filter, how many projects remained in the study. E.g., Initially XXX projects were retrieved from Crates.io. After the cloning locally ZZZ projects remained in the dataset. And so on, and so forth.

The empirical analysis was based on data collected from the Cargo ecosystem, the package ecosystem associated with the Rust programming language. Cargo was selected because it provides a large set of packages with publicly accessible source code repositories and a clearly identifiable ecosystem structure. In addition, Cargo supports the study of both project-level and ecosystem-level knowledge concentration, since many contributors are active in more than one crate.

Commented [EC22]: One might argue that other ecosystems have the characteristics described here. What makes Cargo so special?

The data collection process began by identifying Cargo packages and linking them to their source code repositories. As of May 11, 2026, the Cargo ecosystem contained 265,647 crates.

In order to focus on repositories with sufficient development history for Truck Factor estimation, the dataset was first filtered to include only crates that had been active for more than one year. After applying this criterion, 52,932 crates remained.

Because it was not computationally feasible to apply the analysis to all remaining crates, a sampling strategy was adopted. From these 52,932 crates, the study focused on a subset of 100 repositories, selected on the basis of their dependency importance within the ecosystem. More specifically, the sample focused on highly depended-upon packages, since these are more likely to represent ecosystem-critical components. This choice made the analysis manageable in scope while preserving the ecosystem relevance of the selected repositories.

For each selected crate, the repository URL and project metadata were retrieved from Crates.io. Only packages with accessible version-control repositories and sufficient development history were considered eligible for analysis. Projects were excluded when the repository was unavailable, when the history could not be retrieved, or when the repository structure did not allow reliable extraction of file-level contribution data. After this filtering and repository retrieval process, the associated repositories were cloned locally. At the end of this stage, 80 repositories remained in the final dataset.

For each selected repository, the study collected the information required to apply Avelino et al.'s algorithm such as files contained in the project, the commit history for each file, the identity of the developers who modified each file, and the sequence of modifications over time. This made it possible to determine which developer created a file, how many changes each developer made to it, and how many changes were made by other developers, which are the key inputs used by the Degree of Authorship model which is used by the algorithm of Avelino et al..

Commented [EC23]: Make it clearer that Avelino's code was used to find TF developers.

Because repository data may contain inconsistencies in developer identity because developers may use multiple identities, a preprocessing step was applied to resolve aliases. The same person may appear under different usernames, names, or email addresses in version-control history, and failing to unify these identities may artificially split contributions that belong to one developer. Therefore, whenever possible, developer aliases were merged. Although alias resolution cannot guarantee perfect identification in all cases, it improves the reliability of file ownership estimation.

Commented [EC24]: incorrect phrasing. developers use multiple identities. It's not that repositories create inconsistencies. rephrase.

A further preprocessing step concerned the selection of files to be included in the analysis. Not all files in a repository are equally relevant for Truck Factor estimation. Following the literature, the study distinguished between source files and other repository artifacts such as generated files, vendored dependencies, documentation, configuration files, and test files. Since one of the main objectives of the thesis is to examine the sensitivity of the algorithm to file selection, multiple file-selection configurations were prepared and provided as input to Avelino et al.'s algorithm. These configurations allowed the study to compare the baseline result with alternative estimates obtained when broader or narrower sets of files were considered.

Commented [EC25]: make it more clear that this is handled by Avelino et al's tool.

At the end of the data collection and preprocessing stage, the final dataset consisted of 80 Cargo projects. These repositories formed the basis for both the Truck Factor estimation and the later ecosystem-level analysis.

Commented [EC26]: Given the size of Cargo (265,647 as of May 11), remaining only 80 projects after the filters you mention is not possible/reasonable/etc. You need to mention *ALL* the filters you applied.

3.3 Application of Avelino's Algorithm

To estimate the Truck Factor of each selected project, this thesis adopts the algorithm proposed by Avelino et al. [1]. This algorithm was selected because it is one of the most widely used and empirically supported approaches in the literature [14], and because it provides a clear and reproducible procedure for identifying key developers through file authorship.

Commented [EC27]: applies? where ? I suggest using adopts.

Commented [EC28]: how do we know this? provide a citation?

The algorithm operates in two main phases. In the first phase, it estimates authorship at file level using the Degree of Authorship (DOA) model. The DOA model assigns a score to each developer-file pair based on the developer's relationship to that file. In particular, it considers whether the developer was the original creator of the file, how many times that developer subsequently modified it, and how many modifications were made by other developers. In this way, the model gives higher authorship scores to developers who are more strongly associated with the evolution of a given file.

Commented [EC29]: The actual formulas? I thnk they should be included. An then you can explain that the developer's modifications contribute positively to DOA, while modifications by other contribute negatively.

Once the DOA values are computed, the algorithm identifies the authors of each file by applying threshold rules. In the original approach, raw DOA values are normalized and compared against predefined thresholds so that only developers with sufficiently strong

authorship are considered actual file authors. This step is important because it transforms contribution history into a discrete notion of ownership. However, it is also one of the main sources of methodological sensitivity, since changing the thresholds may alter which developers are considered most knowledgeable for each file.

In the second phase, the algorithm estimates the Truck Factor through an iterative removal process. First, developers are ranked according to the number of project files for which they are identified as most knowledgeable. Then, the top-ranked developer is removed, followed by the next one, and so on. After each removal, the algorithm checks how many files remain covered by at least one remaining author. A file is considered abandoned when all of its identified authors have been removed. The Truck Factor of the project is the minimum number of removed developers required for the proportion of abandoned files to exceed the danger threshold used by the algorithm. In Avelino's original formulation, the project is considered endangered when more than 50% of the relevant files become abandoned.

Commented [EC30]: as authors or as most knowledgable?

Commented [EC31]: I'm not sure I understand this.

In this thesis, Avelino's algorithm was first applied using its baseline configuration to obtain a reference Truck Factor value for each project. The baseline configuration follows the original logic of the method as closely as possible, including the use of DOA-based file authorship, iterative greedy removal of top authors, and the default abandonment rule. The resulting Truck Factor values provide the starting point for the later stability analysis.

After the baseline estimation, the algorithm was applied repeatedly under alternative configurations. These configurations were designed to test how sensitive the final Truck Factor value is to changes in the assumptions of the method. The varied elements included:
(1) the threshold used to determine file authorship from DOA values (normalized_DOA)
(2) the threshold used to determine developer coverage on a Project (TfCoverage)
(3) the selection of project files considered relevant for estimation.

For each project and each configuration, the algorithm produced the following, the estimated Truck Factor value, the set of developers identified as Truck Factor developers, and the number of files covered by those developers. These outputs enabled both a numerical comparison of Truck Factor values across settings and an examination of whether the same contributors remained critical under different methodological assumptions.

In addition to estimating Truck Factor at project level, the study also tracked the identified Truck Factor developers across the ecosystem. When a developer appeared as a Truck Factor developer in more than one project, this was recorded as evidence of ecosystem-level concentration of knowledge. Furthermore, for projects in which the original Truck Factor developers later became inactive, the study examined whether the project continued to survive and, if so, which developers emerged as new Truck Factor developers. This allowed the analysis to go beyond static estimation and explore the maintenance continuity of projects after knowledge loss.

3.4 Stability Assessment Approach and Evaluation Metrics

The main methodological objective of this thesis is not only to estimate Truck Factor, but also to examine how stable that estimate remains when the algorithm parameters are changed. For this reason, the study adopts a stability assessment approach based on repeated execution of the algorithm under multiple configurations and systematic comparison of the results.

Commented [EC32]: estimating TF is not a methodological concern. Remove/rephrase.

Stability in this context refers to the extent to which the Truck Factor estimate of a project remains stable, or changes only slightly, when reasonable variations are introduced into the algorithm. If small methodological changes lead to substantially different Truck Factor values, then the estimate may be considered sensitive to the chosen parameters. Conversely, if the results remain similar across configurations, then the estimate may be considered more robust. To assess stability, the baseline Truck Factor value of each project was compared with the values obtained under the alternative configurations. The study examined whether the Truck Factor value remained identical, increased, or decreased under each parameter setting.

Commented [EC33]: better word to use is stable?

Several evaluation metrics were used for this comparison. First, the study measured the absolute change in Truck Factor, that is, the numerical difference between the baseline estimate and the estimate produced under an alternative configuration. This metric captures the magnitude of change for each project. Second, the study measured the relative frequency of changed estimates, namely the proportion of projects whose Truck Factor value differed from the baseline under each configuration. This provides a direct indicator of how often the algorithm is affected by the tested parameter variation.

Third, the study examined the stability of Truck Factor developers, that is, whether the same developers remained identified as critical when the parameter settings changed. This is important because even when the numerical Truck Factor remains the same, the set of developers considered essential may differ. In such cases, the metric may appear stable numerically while still being unstable in terms of who is identified as holding the critical knowledge.

To quantify the stability of the identified Truck Factor developers, the study compares the developer sets produced under two different setting using a set-overlap measure. For two setting S1 and S2 stability is defined as:

stability(S1, S2) = | TF(S1) ∩ TF(S2) | / | TF(S1) ∪ TF(S2) |

Where TF(S1) and TF(S2) are the sets of developers identified as Truck Factor contributors under the two settings. A value of 1 means that the same developers are identified in both cases, while a value of 0 means that the two sets are completely different. For example, if under one setting the identified developers are {John Doe, Jane Doe, Jimmy Smith} and under another they are {John Doe, Jane Doe}, then the stability is 2/3=0.67. This measure makes it possible to evaluate not only whether the Truck Factor value changes, but also whether the identity of the critical developers remains stable.

Fourth, the study analyzed the distribution of Truck Factor developers across the selected crates from the Cargo ecosystem. For this purpose, the number of projects in which each developer was identified as part of the Truck Factor was recorded. Developers appearing as Truck Factor developers in multiple projects were treated as indicators of ecosystem-level knowledge concentration. This made it possible to estimate not only how many projects were vulnerable individually, but also whether the same individuals represented shared points of failure across the ecosystem.

The results of the stability assessment were interpreted comparatively rather than through a single pass/fail threshold. In other words, the goal was not to label the algorithm as simply stable or unstable, but to identify which parameters produced the largest changes, and whether the observed instability was limited to a small subset of projects or visible across the dataset.

Commented [EC34]: 0.4 Efi, Eleni, Andreas
0.5 Eleni, Andreas
0.6 Eleni, Andreas
0.7 Eleni

stability(t1,t2) = |TF(t1) intersection TF(t2)| / |TF(t1) union TF(t2)|

stability(0.4,0.5) = 2 / 3 = 0.66

Commented [EC35R34]: John Doe, Jane Doe, Jimmy Smith

Commented [EC36]: Given that only 80 projects were analyzed, this statement needs to be downsized.

This approach is appropriate because Truck Factor estimation is inherently approximate, and the purpose of the study is to understand the extent and nature of that approximation.

Taken together, this methodological design makes it possible to assess both the internal stability of Avelino's Truck Factor algorithm and the external significance of its results when applied to a software ecosystem. By combining repository mining, repeated estimation, and ecosystem-level developer analysis, the study provides a structured basis for evaluating how parameter choices influence Truck Factor estimation and what those estimates reveal about knowledge concentration risk in an ecosystem.

3.5 Research Questions

Based on the objective of this study, the thesis is guided by two main research questions.

RQ1: To what extent do Truck Factor estimates remain stable over time and under different parameter settings when Avelino's algorithm is applied to repositories in the Cargo ecosystem?

RQ2: To what extent does the significance of Truck Factor developers remain limited to repository level, and to what extent does it extend to the broader Cargo ecosystem through shared contributors and dependency relationships?

These questions reflect the two main dimensions of the study. The first concerns the methodological stability of Truck Factor estimation, including its temporal behavior and its sensitivity to algorithmic thresholds. The second concerns the ecosystem-level interpretation of the results, especially whether critical contributors represent only project-level risks for interconnected repositories within Cargo.

Section 4

Results and Analysis

4.1 Overview of the Selected Projects

This chapter presents the results of the empirical analysis conducted on the selected repositories from the Cargo ecosystem. The final dataset used for this analysis includes 80 repositories which were examined across four timelines. For each repository and timeline, the Truck Factor was estimated by identifying the developers who appeared as Truck Factor contributors according to the applied methodology.

The selected repositories represent important components of the Cargo ecosystem since they correspond to widely used crates with ecosystem relevance. Their inclusion allows the study to examine knowledge concentration not only within individual projects but also across a broader software ecosystem. Because these repositories differ in size, maturity, contributor activity and maintenance history, they provide a suitable basis for observing variation in Truck Factor estimates across projects and overtime.

The analysis was carried out over four timelines in order to examine whether the estimated Truck Factor remains stable as project history evolves. This longitudinal perspective is important because knowledge concentration is not static. As contributors join, leave or shift their activity, the set of developers considered critical to a project may also change. Therefore,

the results presented in this chapter focus both on the estimated Truck Factor values themselves and on the extent to which the corresponding Truck Factor developers remain stable across time.

Overall, in this chapter we first present the estimated Truck Factor results across the four timelines, then examine the distribution of Truck Factor developers across the Cargo ecosystem and finally discuss the observed temporal stability of the estimates and the algorithm's sensitivity to its parameters.

4.2 Truck Factor Estimation Results

This section presents the estimated Truck Factor results for the 80 repositories across the four timelines which accordingly represent the 1st quarter of the project's lifetime, half of the project's lifetime, the first 3 quarters of the project's lifetime and finally the projects lifetime until today. Table 4.2 summarizes the descriptive statistics of the number of Truck Factor developers identified per repository.

Table 4.2: Summary statistics of estimated Truck Factor values across the four timelines

TF metric	Timeline 1	Timeline 2	Timeline 3	Timeline 4
Repo_Count	80	80	80	80
Mean	3.10	3.14	3.18	3.14
Median	2	2	2	2
Minimum	1	1	1	1
Maximum	15	15	15	15
Standard Deviation	3.10	3.23	3.27	3.29

The results above show that the average estimated Truck Factor remains close to 3 across all four timelines ranging from 3.10 in Timeline 1 to 3.18 in Timeline 3. At the same time, the median remains stable at 2 in every timeline. This indicates that for a typical repository in the dataset, critical knowledge is concentrated in a relatively small number of developers.

However, the distribution of the values shown in Figure 4.2, shows that this pattern is not

Commented [EC37]: remind the reader what is each timeline (either in the text above, or in the column titles).

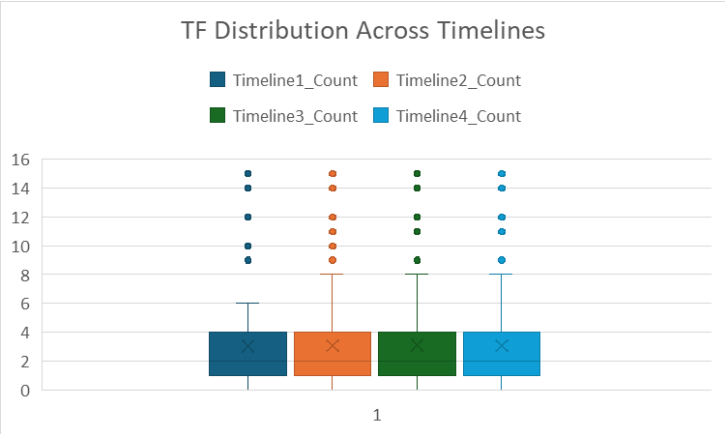
Commented [EC38]: Is this consistent with related work?

Commented [EC39]: what is the spread of values? where to we see it? Say in which figure we can see the distribution.

Commented [EC40R39]: replace spread with distribution.

uniform across all repositories. In every timeline the minimum value is 1, indicating that the most vulnerable repositories in the dataset depend on a single TF developer while the maximum value reaches 15, showing that a small number of repositories distribute critical knowledge across a much larger set of contributors. The relatively high standard deviation in all timelines further confirms that the dataset includes repositories with very different knowledge concentration profiles.

Figure 4.2: Truck Factor Distribution Across Timelines



A particularly notable finding is that repositories with very low Truck Factor values are common in the dataset. Repositories with a Truck Factor of 1 appear frequently in every timeline showing that many projects depend heavily on a single critical contributor. This suggests that knowledge concentration remains a substantial risk for a large portion of the selected Cargo repositories even though the average across the dataset appears moderate.

At the same time, the fact that the average remains close to 3 across all timelines suggests that the overall level of knowledge concentration at ecosystem level is fairly consistent over time. In other words, although some individual repositories show some changes in the TF estimation, the aggregate picture across all the selected repositories remains relatively stable. This means that the selected crates from the Cargo ecosystem contain a persistent pattern in which most repositories rely on a small number of key developers while a smaller subset distributes important knowledge more broadly.

Commented [EC41]: change what? this sentence is not clear.

Commented [EC42]: unclear what this means. What is the aggreate picture? rephrase

Commented [EC43]: as we discussed, you cannot claim anything about the Cargo ecosystem given that only 80 projects were analyzed.

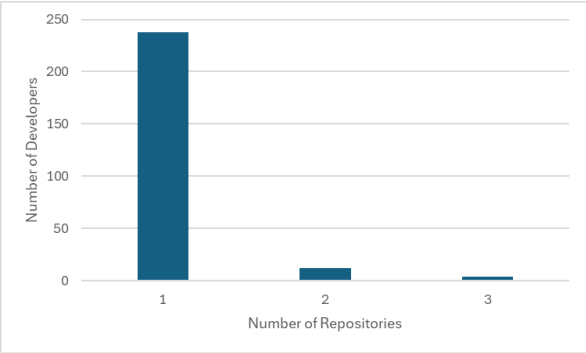
These findings support the view that Truck Factor estimation provides a useful high-level indicator of repository-level knowledge concentration. But the large variation between repositories shows that aggregate statistics alone are not sufficient. A more detailed analysis is needed in order to understand how Truck Factor developers are distributed across repositories and whether the same contributors appear repeatedly in different projects.

4.3 Truck Factor developer distribution across Cargo

Beyond the numerical Truck Factor values, it is also important to examine how identified Truck Factor developers are distributed across the Cargo ecosystem. If the same developers repeatedly appear as Truck Factor contributors in multiple repositories, this may indicate ecosystem-level concentration of knowledge rather than only project-level concentration.

This ecosystem perspective is especially important in the present study because the selected repositories are not arbitrary Cargo projects, but packages on which many other repositories depend on. As a result, the significance of a Truck Factor developer is not limited to whether that developer appears in more than one selected repository. Even when a developer is identified as a Truck Factor contributor in only one of the analyzed packages, their departure may still have wider consequences across the ecosystem if that package is a dependency for many downstream projects. In such cases, knowledge concentration in a single highly dependent-upon repository may create indirect risk for a much larger number of repositories that rely on it for maintenance, updates or compatibility.

Figure 4.3 Developer distribution across the selected repositories



The results in Figure 4.3 show that most Truck Factor developers appear in only one repository, suggesting that in many cases critical project knowledge remains localized within individual projects. However, in the context of this study localization at project level does not necessarily imply limited ecosystem impact, since many of the selected repositories serve as dependencies for numerous downstream projects. They also reveal a smaller set of developers who appear as Truck Factor contributors in more than one repository. These cases are especially important because they point to shared critical contributors whose departure could affect several repositories at once.

Commented [EC44]: where do we see these results?

In the dataset, the most frequently recurring Truck Factor developers appear in three repositories each. These include 4 different developers. In addition, another 12 different developers appear as Truck Factor developers in two repositories. Although the number of highly repeated contributors is not large, their presence is still significant because it shows that some ecosystem knowledge is concentrated across repository boundaries.

Commented [EC45]: A table summarizing these results would be beneficial.

This finding is important from an ecosystem perspective. A repository-level Truck Factor estimate may suggest that the knowledge risk of a project is limited to its own contributor set. However, ecosystem-level risk extends beyond an individual crate not only when the same developer appears as critical in multiple repositories, but also when a developer is critical in a single repository that serves as an important dependency for many others. In such cases, the loss of a single contributor could have consequences for several projects simultaneously, either directly through shared authorship or indirectly through dependency relationships.

At the same time, the relatively limited repetition of Truck Factor developers across repositories suggests that the selected repositories from the Cargo ecosystem do not appear to be dominated by a very small set of universally critical contributors. Instead, the results indicate a more mixed pattern. Knowledge concentration is often strong at the project level while ecosystem-level concentration exists but appears to be limited to a smaller group of cross-project contributors. This is an important distinction because it suggests that the ecosystem's vulnerability may stem both from isolated repository-level dependence and from a modest number of shared critical developers.

Commented [EC46]: remember that 80 projects were analyzed so you should be careful with ecosystem wide statements.

Overall, the distribution of Truck Factor developers across Cargo suggests that ecosystem-level knowledge concentration is present but not extreme. The findings therefore highlight the value of complementing project-level Truck Factor estimation with ecosystem-level analysis, since it's the only way we can reveal whether the same developers act as shared points of failure across multiple repositories and whether critical contributors in highly depended-upon packages create indirect risk for a wider set of downstream projects.

4.4 Stability Analysis of Truck Factor Estimates

In this section, we examine how stable the estimated Truck Factor values remain across the four timelines. Stability here refers to whether the estimated number of Truck Factor developers for a repository remains the same over time or whether it changes as the project evolves.

A first important observation is that the aggregate statistics remain highly consistent across the four timelines. As shown earlier in Table 4.2 the mean, median, and maximum values change only slightly. This suggests that at the dataset-level, the overall pattern of knowledge concentration remains stable. In other words, the Cargo ecosystem continues across the observed timelines to be characterized by repositories that typically depend on a small number of key developers. At the repository level, however, the results show that stability diverges from the dataset-wise stability. Out of the 80 repositories, 61 retain the same Truck Factor value across all four timelines, while 19 show at least one change. A closer examination of these cases shows that the observed instability is usually limited in magnitude, 14 of the changing repositories differ by only one Truck Factor developer across the four timelines, 3 differ by two, and only 2 repositories exhibit larger shifts with maximum differences of 4 and 5 respectively. This indicates that most repositories are numerically stable over time, but a non-negligible minority experiences some degree of change in the size of the identified Truck Factor set. It is important to also note that even a numerically small change may be substantively important when the original Truck Factor is low. For example, a reduction from 2 to 1 Truck Factor developers represents a much stronger increase in concentration risk than the same absolute change in a project whose Truck Factor is already high.

Commented [EC47]: is one change descriptive enough? if it is one out of 2 it's a big deal, if it is one out of 100 it is not that important.

A comparison between the first and the fourth timeline, which represents the 1st quarter of the project's lifetime and the full lifetime of the project, further supports this conclusion. For 65 repositories (out of 80 repositories), the estimated Truck Factor value in Timeline 4 is the same as in Timeline 1. By contrast, 6 repositories show an increase and 9 repositories show a decrease between the first and the last timeline. This pattern suggests that although TF remains stable across repositories, individual repositories may still experience shifts in knowledge concentration over time.

Some repositories show especially large changes. For example, polkadot-sdk [17] increases substantially across the timelines, moving from a Truck Factor of 10 in Timeline 1 to 15 in Timelines 3 and 4. This may indicate broader knowledge distribution over time or a growing number of developers identified as essential under the applied method. Similarly, repositories such as coreutils [18] and holochain [19] also show increases across timelines. By contrast, repositories such as actix-web [20], probe-rs [21], router [22] and rpgp [23] show reductions at particular points in time, indicating stronger concentration of critical knowledge or the loss of previously identified Truck Factor contributors.

The results also show that numerical stability does not necessarily imply full structural stability. More specifically, even in repositories where the total number of Truck Factor developers remained unchanged, the identified TF developers may change over time. Six of these cases appeared in our dataset where the TF count remained stable, but there was internal developer set change. This might happen due to internal redistribution of critical knowledge. Such cases are important because they suggest that purely numerical comparisons may underestimate the extent of change occurring inside the contributor structure of a project.

Commented [EC48]: the fact that you refer to timelines seems a bit vague for the reader here. I don't remember what timeline 4 is... it might be good to rephrase or remind the reader.

Commented [EC49]: how much?

Commented [EC50]: what global picture? remove/rephrase this.

Commented [EC51]: which shifts are meaningful?

Commented [EC52]: what is visible? large changes

Commented [EC53]: provide in a footnote the link to the github repository of this crate. do this for every crate you mention from now onwards.

Commented [EC54]: how many such cases exist? overall, in all the results mentioned above, you need to mention more concrete numbers.

Table 4.4: Summary statistics of Truck Factor developer set stability across timeline comparisons

Comparison	Mean	Median	Minimum	Maximum	St. Deviation	Repos with stability=1
T1-T2	0.89	1.00	0.00	1.00	0.24	62
T2-T3	0.90	1.00	0.00	1.00	0.26	68
T3-T4	0.98	1.00	0.50	1.00	0.08	75
T1-T4	0.84	1.00	0.00	1.00	0.29	57

To complement the numerical comparison of Truck Factor values, the study also measured the stability of the identified Truck Factor developer sets between timelines using the set-overlap metric defined in Section 3.4. Table 4.4 summarizes the resulting stability values across the timeline comparisons. The results indicate a generally high degree of structural stability over time (see the second column of Table 4.4), especially between adjacent timelines. The number of repositories with stability equal to one (see last column of Table 4.4), shows that for at least 71% of the repositories, the same Truck Factor developers were retained across the compared timelines, confirming that adjacent timeline comparisons are highly stable for most projects. By contrast, the comparison between T1 and T4 shows lower structural stability, with a mean value of 0.84, a larger standard deviation (0.29), and exact stability in 57 repositories. The minimum value also dropped to 0.00 for T1-T2, T2–T3 and T1–T4, indicating that in a small number of repositories the Truck Factor developer sets changed completely. Overall, these results suggest that developer-set stability is generally high in the short term but becomes weaker over longer temporal distances as repositories evolve.

Another important pattern concerns repositories with persistently low Truck Factor values. A substantial number of repositories remain at Truck Factor = 1 throughout much of their observed history. These cases indicate enduring dependency on a single critical developer and therefore sustained vulnerability. From a software maintenance perspective, this is one of the most significant findings of the study, since it suggests that for many repositories their knowledge base is not substantially diversified over time.

Commented [EC55]: it's not up to the projects to diversify their knowledge base. Rephrase here.

Overall, the results suggest that Truck Factor estimation is fairly stable at the aggregate level but less uniformly stable at the repository level. The broad ecosystem pattern remains largely unchanged across the four timelines, yet individual projects may still gain or lose Truck Factor contributors as development evolves.

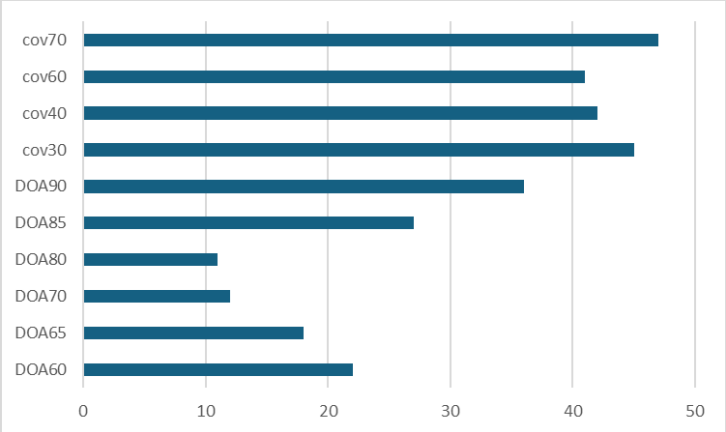
4.5 Sensitivity Analysis of the Algorithm's Parameters

This subsection examines whether the estimated Truck Factor remains stable when the main threshold parameters of the algorithm are varied. More specifically, this analysis evaluates the effect of changing the project coverage threshold and the normalized Degree of Authorship

(DOA) threshold on the Truck Factor results obtained under the baseline configuration.

To assess parameter sensitivity, the baseline results were compared with alternative executions of the algorithm using different threshold values. For the coverage threshold the analysis considered the values 0.3, 0.4, 0.6, and 0.7. For the normalized DOA threshold, the analysis considered the values 0.60, 0.65, 0.70, 0.80, 0.85, and 0.90. The resulting Truck Factor estimates were then compared against the baseline results for all repositories and all four timelines.

Figure 4.5.1: Repositories with changed Truck Factor count under alternative parameter settings



The results in figure 4.5.1 show that varying the coverage threshold had a substantial effect on the final estimates. Depending on the value used, the Truck Factor count changed in 41 to 47 repositories, while the set of Truck Factor developers changed in the same number of repositories. More specifically, the TF count and developer set changed in 45 repositories for coverage = 0.3, 42 repositories for coverage = 0.4, 41 repositories for coverage = 0.6, and 47 repositories for coverage = 0.7. This indicates that Truck Factor estimation is highly sensitive to the coverage threshold in the present dataset.

Varying the normalized DOA threshold also influenced the results, although the effect was less pronounced overall than that of the coverage threshold. The impact was relatively limited for

Commented [EC56]: fix all the commas in the thesis. I will stop correcting from this point onwards.

Commented [EC57]: Comment for all figures/tables in the thesis: if the figure or table is not referred in the text, then it should be removed. If it is included in the text, it needs to be described/explained in the text.

Commented [EC58]: the pictre must be right below its legend, not in the next page.

Commented [EC59]: You need to actually show the results before m mentioning what the results show.

values close to the baseline but became stronger as the threshold moved further away. For example when the DOA threshold was set to 0.70, the Truck Factor count changed in 12 repositories, while at 0.80 it changed in 11 repositories. However, when the threshold was reduced to 0.60 the number of repositories with a changed Truck Factor count increased to 22, and when it was raised to 0.90, the number increased further to 36. This shows that Truck Factor estimation is fairly stable under moderate DOA variation but becomes more sensitive under more extreme threshold settings.

The direction of these changes is also consistent with the logic of the algorithm. Lower coverage thresholds (0.3 and 0.4) produced higher Truck Factor counts than the baseline, whereas higher coverage thresholds (0.6 and 0.7) produced lower Truck Factor counts. This is expected because lower coverage thresholds require more developer removals before the project is considered endangered, while higher thresholds allow the danger condition to be reached sooner. A similar but more gradual pattern appears for the normalized DOA threshold. Lower DOA thresholds (0.60, 0.65 and 0.70) generally produced higher Truck Factor counts than the baseline, while higher DOA thresholds (0.80, 0.85 and 0.90) generally produced lower counts. This is also expected since lower DOA thresholds allow more developers to qualify as file authors, whereas higher thresholds make authorship more restrictive and tend to concentrate ownership in fewer contributors.

Table 4.5: Summary table of sensitivity analysis results

Parameter variation	Repositories with changed TF count	Repositories with changed TF developer set
Coverage = 0.3	45	45
Coverage = 0.4	42	42
Coverage = 0.6	41	41
Coverage = 0.7	47	47
DOA = 0.60	22	23
DOA = 0.65	18	19
DOA = 0.70	12	12
DOA = 0.80	11	13

DOA = 0.85	27	30
DOA = 0.90	36	39

A similar pattern appears when examining the identities of the developers identified as Truck Factor contributors. As we can see in table 4.5, for the coverage threshold, the set of Truck Factor developers changed in exactly the same number of repositories as the TF count, ranging from 41 to 47 repositories depending on the selected value. For the normalized DOA threshold, the set of Truck Factor developers changed in 12 repositories for DOA = 0.70, 13 repositories for DOA = 0.80, 23 repositories for DOA = 0.60 and 39 repositories for DOA = 0.90. This indicates that identity-level sensitivity, especially for the DOA threshold, is slightly stronger than numerical sensitivity. In other words, even when the total Truck Factor value remains unchanged the developers considered critical may still differ under alternative DOA settings.

Commented [EC60]: this needs to be represented in a better visual way for the reader. witha figure or table?

Table 4.5.2: Summary statistics of Truck Factor developer set stability under alternative parameter settings

Commented [EC61]: Comment for all figures/tables in the thesis: if the figure or table is not referred in the text, then it should be removed. If it is included in the text, it needs to be described/explained in the text.

Parameter setting	Mean	Median	Minimum	Maximum	Standard Deviation	Repos with stability = 1 in all 4 Timelines
Cov = 0.3	0.78	0.75	0.33	1.00	0.21	148
Cov = 0.4	0.87	1.00	0.50	1.00	0.16	180
Cov = 0.6	0.86	1.00	0.40	1.00	0.18	181
Cov = 0.7	0.75	0.67	0.20	1.00	0.24	143
DOA = 0.60	0.96	1.00	0.40	1.00	0.10	271
DOA = 0.65	0.97	1.00	0.40	1.00	0.09	282
DOA = 0.70	0.99	1.00	0.50	1.00	0.06	298
DOA = 0.80	0.98	1.00	0.50	1.00	0.08	295
DOA = 0.85	0.92	1.00	0.40	1.00	0.15	231
DOA = 0.90	0.87	1.00	0.33	1.00	0.18	194

Figure 4.5.3: Algorithm's stability under different normalized DOA settings

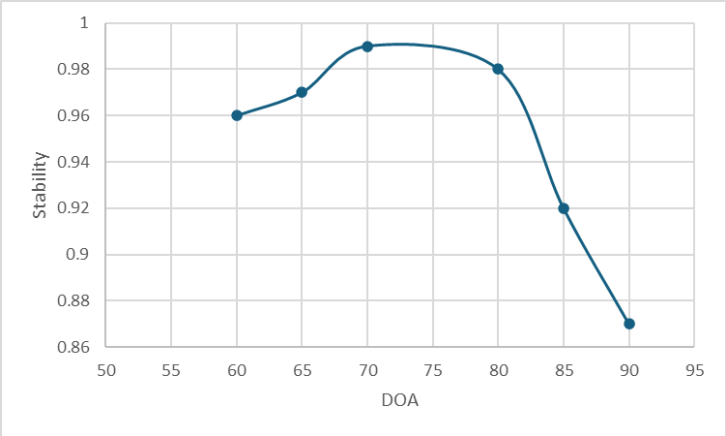
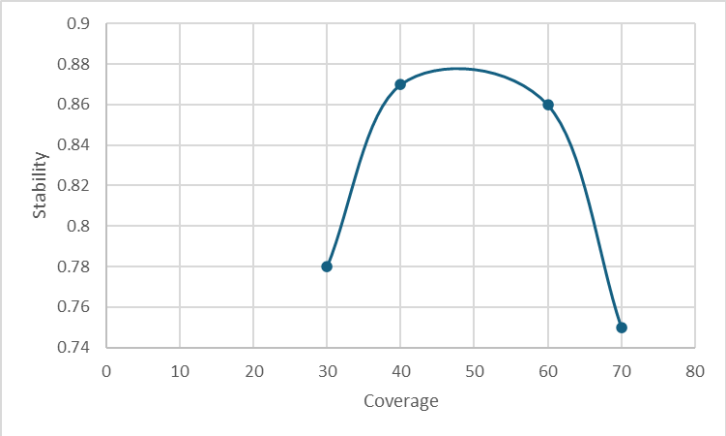


Figure 4.5.4: Algorithm's stability under different Coverage settings



To complement the comparison of changed Truck Factor counts and developer sets, the study also measured the stability of the identified Truck Factor developers between the baseline configuration and each alternative parameter setting using the set-overlap metric defined in Section 3.4. Table 4.5.2 summarizes the resulting stability values. The results show a clear difference between the two types of parameter variation. For the coverage threshold, the mean developer-set stability relative to the baseline ranged from 0.75 to 0.87, with the lowest values observed for Coverage = 0.7 and Coverage = 0.3 (see the 1st column of Table 4.5.2 and Figure

4.5.4). By contrast, Coverage = 0.4 and Coverage = 0.6 showed somewhat higher stability, although both still displayed substantially lower overlap with the baseline than most DOA settings. For the normalized DOA threshold, the corresponding mean stability values were consistently higher, ranging from 0.87 to 0.99 (see Figure 4.5.3). The highest stability was observed for DOA = 0.70 (mean = 0.99, median = 1.00, min = 0.50, SD = 0.06) and DOA = 0.80 (mean = 0.98, median = 1.00, SD = 0.08), while the more extreme values DOA = 0.85 and DOA = 0.90 produced lower overlap. Exact stability (1.00) was also much more frequent for DOA settings, ranging from 194 to 298 cases out of 320, compared with 143 to 181 cases for coverage settings (see the last column of Table 4.5.2). Taken together, these results confirm that alternative coverage thresholds produce much larger structural deviations from the baseline than most DOA variations.

This distinction is important for the interpretation of the metric, because a stable Truck Factor value does not necessarily mean that the same developers remain essential. In some projects, the number of identified Truck Factor contributors remained constant while one contributor was replaced by another. Therefore, the stability of the numerical estimate alone may conceal meaningful changes in the underlying ownership structure of a repository.

Overall, the results suggest that both types of parameter variation tested in this study affect Truck Factor estimation, but the coverage threshold has the strongest impact in the present dataset. While the normalized DOA threshold produces moderate to substantial changes depending on the selected value, the coverage threshold consistently changes the results in roughly half of the repositories. This shows that the estimation process is not equally sensitive to all methodological choices and that the definition of project abandonment plays a particularly important role in the final result.

From a methodological perspective, this finding is important because it shows that Truck Factor estimation depends not only on how authorship is defined, but also strongly on how abandonment is operationalized through the coverage threshold. This means that the interpretation of Truck Factor estimates should pay close attention to both parameters since each can influence the numerical result and the set of contributors identified as carrying critical project knowledge.

Taken together, the sensitivity analysis shows that the baseline results are not fully robust to parameter variation. Although moderate DOA changes often lead to limited differences, the coverage threshold and more extreme DOA values can substantially alter the resulting estimates. As a result, the baseline values presented in the previous sections should be interpreted as meaningful but method-dependent approximations of knowledge concentration.

Section 5

Discussion of Findings

5.1 Discussion and Implications

This thesis set out to examine how stable Truck Factor estimation is when applied to a software ecosystem and to investigate what the resulting estimates reveal about knowledge concentration in the Cargo ecosystem. The findings presented in Section 4 provide several important observations regarding both the interpretation of Truck Factor values and the broader significance of critical contributors in an ecosystem setting.

A first important finding is that the estimated Truck Factor appears relatively stable at the aggregate level. Across the timelines examined the average and median Truck Factor values remain largely unchanged, suggesting that the overall pattern of knowledge concentration in the selected Cargo repositories does not fluctuate substantially over time. This indicates that at ecosystem level, the selected projects consistently rely on a relatively small number of critical developers. In that sense, the results support the usefulness of Truck Factor as a high-level indicator of knowledge concentration risk.

At the same time, the results also show that this aggregate stability should not be interpreted as uniform stability across all repositories. Although most repositories retain the same Truck Factor value across timelines, a non-negligible subset shows increases or decreases in the number of Truck Factor developers. This suggests that project-level knowledge concentration is not static. In some repositories knowledge becomes more distributed over time, while in others it becomes more concentrated. Therefore, one of the central implications of this study is that Truck Factor should not be interpreted only as a fixed property of a project, but also as a measure that may evolve as maintenance responsibilities shift among contributors.

Commented [EC62]: I am missing practical examples of TF (anonymized of course) that show some of the issues you describe/discuss

Commented [EC63]: vague. what is a meaningful subset?

A second important finding concerns the strong presence of repositories with very low Truck Factor values. Many projects in the dataset have a Truck Factor of one or two, meaning that they depend on only one or two critical contributors. This is a significant result from a software maintenance perspective because it highlights the persistence of concentrated knowledge in widely used Cargo packages. Even if the average Truck Factor across the ecosystem appears moderate, the presence of many low-TF repositories suggests that ecosystem resilience may be weaker than aggregate statistics alone imply. In practical terms, a repository with a Truck Factor of one remains highly vulnerable to contributor departure or inactivity.

The ecosystem perspective of this thesis adds an additional layer to the interpretation of these results. Previous Truck Factor studies have mainly focused on individual repositories, but the present study shows that ecosystem-level risk can emerge in more than one way. First, 16 developers appear as Truck Factor contributors in multiple repositories, which means that their departure could affect several projects simultaneously. Second, and equally important, many of the repositories selected in this study are widely depended-upon packages. As a result, even when a Truck Factor developer is critical in only one repository, their departure may still create wider ecosystem consequences if that repository serves as an important dependency for many downstream projects. This means that ecosystem vulnerability does not depend only on shared developers across repositories, but also on the structural importance of the repositories in which those developers are critical. This distinction has important implications for how Truck Factor should be interpreted in software ecosystems. A project-level Truck Factor estimate may suggest that the knowledge concentration risk of a repository is local and self-contained. However, the impact of losing a critical contributor may propagate beyond the repository itself. In other words, the ecosystem significance of a Truck Factor developer depends not only on how many repositories they are critical in, but also on how central those repositories are to the rest of the ecosystem. This reinforces the argument that project-level and ecosystem-level analyses should be used together rather than separately.

The sensitivity analysis further refines the interpretation of Truck Factor estimation. The results show that the estimates are sensitive to variations in both the coverage threshold and the normalized DOA threshold. In the present dataset, the coverage threshold produced the strongest effect, changing the Truck Factor count and developer set in roughly half of the repositories. The normalized DOA threshold also affected the results, especially when more

Commented [EC64]: how many?

Commented [EC65]: by how many packages? I don't recall reading this information.

Commented [EC66]: you cannot start a paragraph with "this distinction ..." what distinction? if it refers to the previous pagraph, then you should not start a new paragraph here.

Commented [EC67]: all ecosystems are dependency heavy. I suggest to remove this part.

extreme values were used, but its impact was generally smaller than that of coverage. This finding suggests that not all methodological choices affect the metric equally and that the definition of project abandonment plays a particularly important role in the final estimate.

An especially important observation is that identity-level instability may occur even when the numerical Truck Factor remains the same. In several repositories, the number of Truck Factor developers did not change under different DOA thresholds, but the actual developers identified as critical did. This suggests that numerical stability alone is not sufficient for evaluating the robustness of the metric. A project may appear stable if only the Truck Factor count is considered while in reality the underlying ownership structure has shifted. For this reason, future studies of Truck Factor stability should consider not only changes in the metric's value but also changes in the set of contributors that the algorithm identifies as essential.

Taken together, the findings of this thesis suggest that Truck Factor remains a useful indicator of knowledge concentration, but one that must be interpreted carefully. It is useful because it captures an important and intuitive dimension of project vulnerability. However, it must be interpreted carefully because the metric is shaped by methodological assumptions, may conceal internal changes in contributor identity, and may underestimate broader ecosystem risk when it is applied only at repository level.

From a practical perspective these results may be valuable for maintainers, project communities, and ecosystem stakeholders. Repositories with persistently low Truck Factor values may benefit from stronger documentation practices, broader code ownership, contributor onboarding, and mentoring of new maintainers. At ecosystem level, packages that are both highly depended upon and strongly knowledge-concentrated may deserve particular attention, since weaknesses in such repositories could affect many downstream projects. Therefore, the findings of this thesis support the idea that Truck Factor can serve not only as an academic metric but also as a practical signal for identifying repositories in need of resilience-improving interventions.

Taken together, the findings provide answers to the two research questions that guided this thesis. With respect to RQ1, the results show that Truck Factor estimation is broadly stable at the aggregate level but not fully stable at repository level and not fully robust to parameter variation. In particular, the estimates were highly sensitive to changes in the coverage threshold

and moderately sensitive to changes in the normalized DOA threshold. With respect to RQ2, the results show that the significance of Truck Factor developers extends beyond repository level. This occurs both when the same contributors appear as critical in multiple repositories and when a contributor is critical in a single repository that serves as an important dependency for many downstream projects. Therefore, the findings support an ecosystem-level interpretation of Truck Factor risk rather than a purely repository-level one.

Overall, this study contributes to the literature by showing that Truck Factor estimation in a software ecosystem is informative at multiple levels. It reveals project-level dependence on key contributors, highlights ecosystem-level patterns of concentrated expertise, and demonstrates that the robustness of the estimate depends more strongly on the coverage threshold than on the authorship-related assumptions tested here. In this sense, the thesis extends the study of Truck Factor from isolated repositories to a broader ecosystem context and provides a more nuanced understanding of what stability means for this metric.

5.2 Threats to Validity

Commented [EC68]: Threats to validity in a study are indeed factors that can affect different types of validity. But it is not enough to just list them. For each threat, we need to report how prevalent this threat is and what was done to minimize its effect. If it wasn't possible to mitigate, you need to discuss to what extent it might make your results unreliable...

As with any empirical study based on repository mining, the findings of this thesis should be interpreted in light of several threats to validity.

A first threat concerns construct validity. That is whether the operationalization used in the study accurately captures the concept of knowledge concentration. The Truck Factor is intended to represent the minimum number of contributors whose loss would place a project in serious difficulty. However, the estimation procedure used here relies on repository history as a proxy for actual project knowledge. Commit activity and file authorship provide useful evidence of developer involvement, but they do not fully capture all forms of knowledge relevant to software maintenance. Important knowledge may also be acquired through code reviews, issue discussions, meetings, documentation, informal mentoring, or architectural decision-making that does not appear directly in version-control data. As a result, the Truck Factor values reported in this thesis should be interpreted as estimates of knowledge concentration based on observable repository activity rather than direct measurements of all knowledge present in a project.

A second construct-related threat concerns the specific choice of Avelino's algorithm [1] and the Degree of Authorship model. Although this approach is well established in the literature, it still reflects one particular way of defining authorship and abandonment. Other Truck Factor formulations may assign ownership differently or use alternative rules for determining when a project becomes endangered. Therefore, the results of this study should not be interpreted as the only possible representation of knowledge concentration in the Cargo ecosystem but rather as the outcome of one widely used methodological approach.

A third threat concerns internal validity, especially with respect to preprocessing decisions. Repository-mining studies depend on data cleaning steps such as alias resolution, file filtering, and extraction of commit histories, and limitations in these stages may affect the final estimates. One such threat is developer identity ambiguity, since the same contributor may appear under multiple names, email addresses, or usernames. To reduce this threat, the study applied alias resolution and merged developer identities whenever possible. In particular, contributor information available through the repository platform was used when accessible, and cases that could not be resolved automatically were inspected and merged manually. Another threat concerns file selection, because including irrelevant artifacts such as generated files, vendored dependencies, documentation, configuration files, or test files may distort the ownership structure of a project. To reduce this effect, the study focused on source files and excluded non-relevant categories following the literature. A further threat concerns repository history extraction, since incomplete or inaccessible histories may lead to missing or distorted contribution data. To address this, only repositories with accessible version-control data and sufficient development history were retained in the dataset, while repositories that could not be cloned or did not support reliable extraction of file-level contribution information were excluded. Although these measures reduce the effect of preprocessing-related threats, they cannot eliminate them completely, and some uncertainty remains in cases of imperfect alias resolution or incomplete historical information.

A fourth threat concerns external validity, or the extent to which the findings generalize beyond the selected dataset. This study focuses on a set of Cargo repositories chosen because of their ecosystem relevance and dependency importance. While this makes them highly suitable for investigating ecosystem-level risk, it also means that the findings may not generalize directly to all Cargo packages, to less central repositories or to software ecosystems with different structures, governance models or contribution cultures. For example ecosystems with different

Commented [EC69]: not sure how ecosystems have organization control. are you talking about governance? rephrase.

dependency patterns or alternative development practices might exhibit different Truck Factor behavior.

Related to this, the study emphasizes widely depended-upon packages which means that the findings are especially relevant for central ecosystem components. This strengthens the practical importance of the results, but it may also bias the sample toward repositories that are already more mature, more visible or more actively reused than typical Cargo projects. Therefore, the results should be interpreted primarily as evidence about ecosystem-critical repositories rather than about the entire population of crates.

A further and final threat to external validity concerns the methodological scope of the sensitivity analysis. While this thesis examined the effect of varying the coverage threshold and the normalized DOA threshold, additional parameters may also influence Truck Factor estimation. These include file-selection policies, the inclusion or exclusion of documentation, test files, generated files, and third-party code, the treatment of developer aliases, the authorship model used, the relative importance assigned to first authorship and subsequent modifications, the granularity of ownership attribution (for example, file-level versus line-level), the definition of file abandonment, and the danger threshold used to determine when a project is considered endangered. Such choices may alter the Truck Factor estimate by changing the distribution of ownership across files, the set of developers recognized as critical, or the stopping condition under which the project is classified as vulnerable. These factors were not exhaustively explored in this thesis because doing so would have considerably expanded the scope of the study beyond what was feasible in terms of time, computational resources, and thesis scale. As a result, the findings should be understood as applying to the methodological assumptions and parameter ranges investigated here, rather than to every possible formulation of Truck Factor estimation.

Despite these limitations, the study provides a useful and systematic analysis of Truck Factor estimation in a software ecosystem. By combining repository-level estimation, ecosystem-level interpretation, temporal comparison and parameter sensitivity analysis, it offers a structured basis for understanding how knowledge concentration appears in the Cargo ecosystem and how stable the corresponding estimates remain under methodological variation.

Section 6

Conclusion

This thesis examined the stability and sensitivity of Truck Factor estimation in a software ecosystem by applying Avelino's algorithm [1] to selected repositories from the Cargo ecosystem. The study focused on two main research questions. Whether Truck Factor estimates remain stable over time and under different parameter settings, and whether the significance of the results remains limited to repository-level or if they extended to the broader selected repositories from the Cargo ecosystem.

The results showed that Truck Factor estimation is relatively stable at the aggregate level. Across the four timelines the average and median Truck Factor values remained largely unchanged, suggesting a consistent overall pattern of knowledge concentration in the selected repositories. At the same time, this stability was not uniform at repository level since some projects showed an increase or decrease in their estimated Truck Factor over time, and even in some cases were there was not any change in the estimated TF count there was visible change in the set of the developers.

The study also found that many repositories have low Truck Factor values often equal to one or two, indicating continued dependence on a small number of critical developers. From an ecosystem perspective this risk is amplified by the fact that the analyzed repositories are widely depended-upon packages. Therefore, even when a developer is critical in only one repository their departure may still affect many downstream projects.

The sensitivity analysis showed that the results were sensitive to changes in both the coverage threshold and the normalized DOA threshold. In particular, the coverage threshold had the strongest impact on the final estimates, while the normalized DOA threshold produced moderate to substantial changes depending on the value used. This suggests that Truck Factor estimation depends strongly on both how authorship is defined and how project abandonment is operationalized, with the coverage threshold showing the strongest effect in the present

dataset. It also showed that the set of identified Truck Factor developers may change even when the numerical Truck Factor remains the same.

Overall, the thesis shows that Truck Factor is a useful but approximate indicator of knowledge concentration risk. Its value lies in highlighting dependence on key contributors but its interpretation should consider both methodological assumptions and the ecosystem context of the analyzed repositories. In this sense, the study contributes to a more complete understanding of Truck Factor by examining it not only as a repository-level metric, but also as an ecosystem-level indicator of software sustainability.

Future work could extend this study in several directions. First, additional sensitivity experiments could examine a wider range of parameter settings, including alternative file-selection policies, different danger thresholds, and other definitions of abandonment, in order to better understand which methodological choices have the greatest influence on Truck Factor estimation. Second, future research could explore alternative ownership models beyond the Degree of Authorship approach, including models that place greater emphasis on recency, review activity, or line-level contribution. Third, the analysis could be expanded to larger samples of Cargo repositories or applied comparatively across different software ecosystems in order to determine whether the patterns observed in this thesis generalize beyond the selected dataset. Finally, future work could incorporate additional sources of developer knowledge beyond version-control history, such as code reviews, issue discussions, documentation activity, and communication records, so that Truck Factor estimation can better reflect the broader socio-technical processes through which project knowledge is created, shared, and maintained.

Bibliography

[1] Avelino et al, “A Novel Approach for Estimating Truck Factors”, 24th International Conference on Program Comprehension, pp. 1-10, 2016

[2] Mathieu Goeminne and Tom Mens, “Evidence for the Pareto principle in Open Source Software Activity”, Joint Proceeding of MDSM 2011 and SQM 2011, pp. 78-86, 2011

[3] P. N. Robillard, “The role of knowledge in software development”, Communications of the ACM, Vol. 42, N. 1, pp. 87-92, 1999.

[4] Bird et al, “Dont touch my Code! Examining the effects of ownership on software quality”, Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering, pp. 4-14, 2011

[5] Fritz et al, “A Degree of Knowledge Model to Capture Source Code Familiarity”, Proceeding of the 32nd ACM/IEEE International Conferene on Software Engineering, V. 1, pp. 385-394, 2010

[6] Jabrayilzade et al, “Bus factor in Practice”, Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice, pp. 97-106, 2022

[7] S. A. Piccolo, P. De Meo, G. Terracina, G. Greco, “The theory and practice of computing the bus factor”, Information Sciences, V. 743, pp. 123353, Elsevier, 2026

[8] T.Mens, CD.Roover, “An introduction to software ecosystems”, Software Ecosystems: Tooling and Analytics, pp. 1-29, 2023

[9] Schueller et al, “Evolving collaboration, dependencies, and use in the Rust Open Source Software ecosystem”, Scientific data 9, 703, 2022

[10] Zazworka et al, “Are developers complying with the process: an xp study”, Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 1-10, 2010

[11] Ricca et al, “On the difficulty of computing the truck factor”, International Conference on Product Focused Software Process Improvement, pp. 337-351, 2011

[12] Cosentino,Izquierdo,Cabot, “Assessing the Bus Factor of Git repositories”, IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering, pp. 499-503, 2015

Commented [EC70]: Cite properly: https://dl.acm.org/doi/10.1145/291469.291476

Commented [EC71]: Fix all citations

[13] Rigby et al, “Quantifying and mitigating turnover-induced knowledge loss: Case studies of Chrome and a project at Avaya”, Proceedings of the 38th International Conference on Software Engineering, pp. 1006-1016, 2016

[14] Ferreira et al, “A comparison of three algorithms for computing Truck Factors”, IEEE/ACM 25th International Conference on Program Comprehension, pp. 207-217, 2017

[15] Cury et al, “Source code expert identification: Models and application”, Information and Software Technology, V. 170, pp. 107445, Elsevier, 2024

[16] https://doi.org/10.1007/s10664-021-10012-6, Last accessed 3/5/2026

[17] https://github.com/paritytech/polkadot-sdk, Last accessed 12/5/2026

[18] https://github.com/uutils/coreutils, Last accessed 12/5/2026

[19] https://github.com/holochain/holochain, Last accessed 12/5/2026

[20] https://github.com/actix/actix-web, Lat accessed 12/5/2026

[21] https://github.com/probe-rs/probe-rs, Lat accessed 12/5/2026

[22] https://github.com/iron/router, Lat accessed 12/5/2026

[23] https://github.com/rpgp/rpgp, Lat accessed 12/5/2026