MASTER'S THESIS 2024

Case Study on Open Source Forks: Impact on Knowledge Distribution and Code Quality

Anja Pettersson, Nova Svensson

ISSN 1650-2884
LU-CS-EX: 2024-32

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2024-32

Case Study on Open Source Forks: Impact on Knowledge Distribution and Code Quality

Fallstudie av förgrening i öppen källkod: påverkan på kunskapsdistribution och kodkvalitet

Anja Pettersson, Nova Svensson

Case Study on Open Source Forks: Impact on Knowledge Distribution and Code Quality

Anja Pettersson
an2714pe-s@student.lu.se

Nova Svensson
no8664sv-s@student.lu.se

June 18, 2024

Master's thesis work carried out at CodeScene AB.

Supervisors: Joseph Fahey, joseph.fahey@codescene.com
Markus Borg, markus.borg@cs.lth.se

Examiner: Alma Orucevic-Alagic, alma.orucevic-alagic@cs.lth.se

Abstract

High-quality source code and effective knowledge distribution are important factors for software projects to be successful. Sometimes Open Source Software undergoes forking, resulting in the creation of independent projects from forked source code. This thesis conducts a case study analyzing how forking affects Open Source Software projects, focusing on knowledge distribution and code quality using CodeScene.

The results showed that three out of four projects experienced prolonged knowledge loss from forking, with no apparent correlation between knowledge loss, code quality changes, bug frequency, and forking. This implies that Open Source communities should be aware of the consequences of forking, that it can lead to extended periods of knowledge loss and challenges in regaining previous knowledge levels. Insights important for communities' decision-making and resource allocation regarding knowledge management and retention. Future studies could investigate how social aspects and governance, leadership and community dynamics affect projects undergoing forking.

Keywords: Open Source Software (OSS), Case Study, Forking, Knowledge Distribution, Code Quality

Acknowledgements

We would like to thank our supervisors Markus Borg and Joseph Fahey for their commitment and support throughout our work with this thesis. We would also like to thank CodeScene for making us feel welcome and providing us with the opportunity to use their tool.

Contents

Chapter 1
Introduction

In the competitive landscape of software development, speed is more than just an advantage - it is a necessity. Investing in efficient development methodologies does not just accelerate time-to-market, it also creates a competitive edge. However, for companies to truly succeed, prioritizing high-quality source code is essential. This approach actively mitigates Technical Debt (TD) - the additional rework cost incurred when opting for quick fixes over more thorough, although time-consuming, solutions. Recent studies show that developers can spend up to 42% of their time resolving issues caused by TD [77].

Another key factor in the success of a software project is effective knowledge distribution. When turnover occurs, i.e., when key developers leave, the resulting knowledge loss is often very costly. Knowledge loss, much like the code quality issues associated with TD, diminishes an organization's capability to preserve and modify its code base effectively. Research has shown that project files could be abandoned for more than two years [53].

This thesis will investigate what potential effect forking has on open source software (OSS) projects in terms of knowledge loss and code quality. Project aim and objectives are further discussed in chapter 3.

OSS refers to software developed and distributed under licenses that comply with the Open Source Definition (OSD) [40]. The non-proprietary nature of intellectual contributions is a fundamental aspect of this concept, facilitating software development and distribution that is transparent, modifiable, and freely accessible [68]. It should be acknowledged that the specific permissions and distribution terms associated with OSS may vary, depending on the particular OSS license agreement that has been implemented [39]).

OSS has seen a rise in both popularity and scale. In "The 2023 State of Open Source Report" by OpenLogic by Perforce and the Open Source Initiative (OSI) [38], they present survey results where 80% of the respondents say the use of OSS in their companies increased, and 41% say the use has increased significantly [7]. A survey done by The New Stack and The Linux Foundation in 2018 shows that 80% of the respondents' companies use OSS for internal, noncommercial reasons, and over 60% use it for commercial reasons[35]. This expansion means that maintaining such projects is not feasible for a single developer; instead,

it demands the collective efforts of many. Consequently causing a high risk of knowledge loss which might lead to increased TD.

These dynamics of OSS projects take an even more complex turn when these projects undergo forking. Forking is the process of creating a new independent project by copying and modifying an existing codebase. When OSS projects are forked, it often results in members from the original project participating in the new project. Therefore, for large projects, the fork is not just a code fork, it is also an organizational fork creating a new organization. All of the post-fork teams have to carry on managing the whole project without the knowledge the other team has. This could potentially cause knowledge loss in the new post-forked project organizations, resulting in decreased code quality. Research shows that there are various motivations as to why forking is initiated in OSS projects, where governance disputes and technical motivations are among the most common [71] [5]. We elaborate on the different types of forking in section 2.2.

The project will be carried out in collaboration with the company CodeScene which provides a software engineering intelligence platform [10]. This platform can be used to identify patterns in the development and maintenance of a system by analyzing the progress of a software development project. It can, for example, give insight into team-code alignment, code quality, help prioritize technical debt, and detect vulnerable distribution of knowledge.

1.1 Motivation and Project Aim

The motivation for this study stems from recognized challenges in OSS projects, where turnover has been identified as having a negative impact on project quality [31], and contributor productivity [8]. This is mainly attributed to knowledge loss which also poses a significant threat to the overall sustainability of the project.

In light of this, this research aims to explore the impact of OSS forking on knowledge distribution and code quality. We conduct two case studies on past incidents of forking, looking at both the original and the forked projects. By investigating the scope and duration of knowledge loss post-forks and examining the impact on code quality, our findings provide valuable insights. In the future, they can help formulate effective strategies for knowledge continuity in OSS projects, and project forks. Ultimately contributing to the maintenance of high-quality code and knowledge retention.

1.2 Previous Work

P.C. Rigby et al [70] conducted a study on knowledge loss within software projects. Their findings revealed that knowledge loss in software projects tends to exhibit a *positively skewed* distribution. They suggest that while most projects commonly experience minor knowledge losses over time, they occasionally encounter significant losses. Building upon this work, Nassif and P.Robilard conducted a replication study [53], arriving at the same conclusions, thus reinforcing the validity of Rigby et al.'s findings. They also found that files left behind by developers leaving a project often remained abandoned for a long time. These studies collectively shed light on the impact of developer turnover within OSS projects [70, 53].

Expanding on this understanding, Izquierdo-Cortazar et al. proposed a method for measuring knowledge loss due to developer turnover, rooted in the concept of software archaeology [43]. They advocate calculating the knowledge loss based on the amount of "orphaned code" left by departing developers using a source code management system. "Orphaned code" refers to code written by developers who left the project. This approach offers valuable insight for mitigating the challenges posed by developer turnover [43].

Furthermore, Gamalielsson and Lundell [33] conducted a study where they investigated the sustainability of forked OSS projects. More precisely, they looked at the possibilities of forked OSS projects being sustainable long-term by investigating OpenOffice.org and its fork LibreOffice. Their research reveals that, despite originating as a fork of the OpenOffice.org project, LibreOffice achieved sustainability by attracting active committers from its predecessor. While the majority of contributors came from the original project, their integration into the new fork ensured continuity of knowledge and work practices, thereby ensuring the project's longevity [33].

Shifting the focus to code quality, Tornhill and Borg conducted a quantitative study where they investigated code quality's business impact in 30 Proprietary Production Codebases [77]. Their analysis reveals that there are as many as 15 times more flaws in code of low quality, than in code of high quality. They also found that when addressing issues in code of low quality the time it takes to resolve the issue takes on average 124% longer. Through these findings, they want to highlight the importance of working for and maintaining high code quality because of the substantial impact it has on business.

Chapter 2
Background

This chapter explains Open Source Software (OSS), how contributing and collaborating on OSS works, how OSS can be used by companies and how knowledge is managed in OSS. Furthermore, forking, open-source licensing and some licenses are described, as well as code quality and a description of metrics in CodeScene.

2.1 Open Source Software

This section describes contributions and collaborations in OSS, the use of OSS by companies, which is called Industrial Open Source, as well as knowledge management in OSS projects.

Contributions and Collaborations in OSS

OSS is characterized by its community-driven approach. Contributions come from various participants, including individual volunteers, academic researchers, and corporate entities [29]. Various tools and platforms facilitate this collaboration, with GitHub emerging as one of the most prominent platforms in recent years [20]. These platforms provide essential tools for distributed and often asynchronous development used during OSS development [75].

Industrial Open Source

Industrial open source refers to companies using open source to create commercial products and innovative business models, replacing traditional proprietary development [29]. Companies across various sectors have widely adopted OSS developed and maintained by external parts in their software products and services today [29, 50]. However, the ways in which companies approach OSS development are diverse. Both company managers and individual developers are involved in deciding if and how a company is involved in community OSS

projects, but most contributions come from developers or employees [73]. Multiple factors can affect an employee's ability to contribute to OSS projects [6]. These can be technical, fiscal, or time constraints, as well as what license is used, the project's governance, and other contributors' strategic interests.

Knowledge Management

It is not unusual for OSS contributors to come and go, or for the contributors to be geographically spread out. Because of this the need for efficient knowledge management is important, or else the risk of knowledge loss is high. To manage this risk, i.e. to manage the knowledge in an OSS project there has to be knowledge sharing. For there to be knowledge sharing, the community and its members have to be motivated and want to share their knowledge [60].

One thing that helps members to share their knowledge is by using a shared language in the community, as well as having social networks and an organizational culture with set values, beliefs, attitudes, systems, and rules [60]. Because of the geographically distributed members, there have to be tools and methods to do this effectively. GitHub is a tool that can be used for this purpose, where information and documentation can be shared. Other ways to help share knowledge is through messaging platforms where members can interact in real-time. Examples of such platforms are Slack, Discord, and Stack Overflow [60].

Regarding how motivated the members are to share their knowledge, one factor is how satisfied they are with the management [60]. If the members are satisfied with the management, they are likelier to contribute to knowledge sharing.

2.2 Forking

The concept of forking in open-source development has significantly evolved, moving from the traditional practice of 'Hard Forks' — duplicating a repository for creating a separate development path under a different name — to the modern practice of 'Social Forks'. Social forks involve making public copies of repositories in distributed version control systems, such as GitHub, with the primary aim of facilitating collaborative changes and potentially integrating these modifications back into the original project [85]. However, for the purpose of this research, the focus will solely be on hard forks.

The motivations behind initiating a hard fork of OSS projects have been the subject of investigation in several studies [59, 71]. However, a majority of these studies were conducted prior to the widespread adoption of GitHub — which has been acknowledged to substantially alter the practices of forking [85]. In 2012, G. Robles and M. Gonzales-Barahona [71] conducted a comprehensive pre-github study on Software Forks, where they investigated the motivation and outcome of 220 projects. More recently, in 2022, Businge et al. [5] undertook a study to further explore the motivations behind creating a fork, confirming that the findings from [71] still hold. Below is a list of motivations behind forks, categorized according to their prevalence, with rankings ranging from the most common to the least common. These classifications and ranking are derived from research findings in [5] and [71].

1. **Technical**: (Diverging Features) When developers face opposition from main contributors when introducing new project functionalities. An example of this type of fork is Poppler [69], a forked project from xpdf [82].

2. **Governance Disputes**: (Diverging Interests) When the leaders overlook the community, and contributors feel that their feedback is ignored or development progress too slow. This prompts the contributors to create a fork for more open development practices. An example of this type of fork is io.js [42] forked from Node.js [58].

3. **Legal**: (Diverging Licences) Conflicts regarding licenses or trademarks, and changes causing issues with adhering to rules and regulations. An example of such a fork is XFree86 [80] and the forked project X.Org [81].

4. **Commercial Strategy**: When a company forks an existing project as a part of a commercial strategy, releasing it as free software or creating a proprietary version. An example of such a fork is LibreOffice fork from OpenOffice.org, as well as Webkit [78] forked from KHTML [49].

2.3 Open-Source Licenses

To establish the terms under which OSS can be utilized, modified, and distributed, various OSS licenses govern these practices [83]. Many open source projects use established licenses, but there are also open source projects who choose to create their own license. There are two primary types of OSS licenses: copyleft and non-copyleft licenses. The copyleft licenses are more restrictive and require that any derivative work, when integrated with other work, must adhere to the copyleft license terms [19].

Serving as a central authority in this domain, the Open Source Initiative (OSI) curates a list of OSI-approved licenses [39]. To attain OSI approval, a license undergoes a thorough review process ensuring its alignment with community norms and the OSI's definition of Open Source [41].

Below is a list of common open-source licenses, including OSI-approved options, alongside specialized and proprietary licenses.

- **ALv2:** Apache License, Version 2.0 (ALv2) is an OSI-approved license that is labeled as "Popular/Strong Community" [39]. For example, under the ALv2 changes made to the OSS can have additional copywriter statements and different terms for distribution, use, and modification of the modified OSS [76].

- **MIT License:** MIT License is another "Popular/Strong Community" OSI-approved license [39]. This license is a permissive license and allows "[...] to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software [...]" [37].

- **AGPL-3.0:** GNU Affero General Public License version 3 (AGPL-3.0) is a copy left license and also OSI-approved [32]. This license is designed to make sure that widely used versions of programs' source code become available for public access.

- **ELv2:** Elastic License 2.0 (ELv2) is classified as a non-copyleft license and is not approved by the OSI. This license allows instances to modify, redistribute, and use with limitations ELv2 permits modifications, redistribution, and use but prohibits providing the products as a managed services [24].

- **ownCloud Commercial License:** ownCloud Commercial License, a non OSI-approved licences, is utilized by ownCloud customers who purchase an Enterprise subscription [66]. This license enables customers to modify code while retaining the confidentiality of their intellectual property [65].

- **SSPL:** Server Side Public License (SSPL), version 1, is a license not approved by the OSI. It stipulates that any modifications or alterations offered as a service to third parties must be accessible for free download by anyone under the SSPL terms [52].

2.4 Code Quality

Code quality is not defined by one thing. It is a multifaceted construct that includes aspects such as maintainability, functionality and reliability [44]. One way that code quality has been measured is by using the code health metric in CodeScene. Tornhill and Borg use CodeScene's Code Health metric as a representation of code quality in their quantitative study investigating code quality's business impact in 39 Proprietary Production codebases [77]. Their analysis reveals that there are, on average, 15 times more defects in code of low quality, than in code of high quality. They also found that when addressing issues in code of low quality, the time it takes to resolve the issue takes on average 124% longer. Through these findings, they want to highlight the importance of working for and maintaining high code quality because of the substantial impact it has on business [77].

Their research shows that code quality has on the overall success and efficiency of a software project. The code health metric, as measured by CodeScene, provides a valuable quantitative assessment of code quality. As Tornhill and Borg's study demonstrates, there is a correlation between Code Health scores and the presence of defects within the code. Therefore, the Code Health metric will be used to study the software quality [77].

In their more recent study about code maintainability, they further investigate how the number of defects relates to code health [4]. One of their findings is that the average number of defects decreases as the code health goes from a score of 1 to a score of 5 where it plateaus. However, for files with a code health score of 8 or more the number of defects decreases again. Another one of their findings is that the probability of files being very big is high for files that have a score of 1 on code health [4].

2.5 Metrics in CodeScene

This thesis utilizes CodeScene as a code analysis and visualization tool. This thesis has used 8 metrics from CodeScene: Code Health, Hotspot files, Knowledge Loss, Knowledge Distribution, Main Author, Author Churn diagram, and issues classed as defects. This section describes what these metrics are and how they are calculated.

Code Health The Code Health score in CodeScene relies on patterns associated with higher maintenance expenses [11]. CodeScene calculates a score ranging from 1 to 10 to evaluate the health of the codebase. A higher score suggests that the code is easy to comprehend and evolve, while a lower score indicates significant quality concerns. This score considers

various aspects of both the code itself and organizational elements. CodeScene evaluates between 25 to 30 factors, depending on the programming language used, to calculate the score. These factors are not disclosed by CodeScene to maintain a competitive advantage. CodeScene sorts files into categories: unhealthy (score 4.0 or lower), problematic (score between 4.0 and 8.0), or healthy (score 8.0 or higher).

In CodeScene two key performance indicators (KPIs) regarding code health are Hotspot Code Health and Average Code Health [9]. Hotspot Code Health is determined by calculating the weighted average of code health within the hotspots (see section 2.5). This metric holds significant importance as it highlights areas where low code health can lead to increased costs. Average Code Health is calculated as a weighted average across all files within the codebase. This provides insight into the extent of any possible code health issues. The score is aggregated by a weighted average to provide a more accurate health score. The weight of each file is determined by the number of lines of code (LoC) it contains. This ensures that files with more LoC, have a greater impact on the overall score compared to smaller files. In this thesis, the term 'code quality' will refer to the code health metric in CodeScene.

Hotspots Hotspots highlight code where there is the most activity and development. Consequently, hotspots pinpoint modules that are likely to have a significant impact on the business and impact the cost of maintenance [11]. Hotspots can be calculated on three levels in CodeScene: file-level, architectural level and method level. File-level and method level hotspots are calculated by CodeScene, while architectural level refers to a user-defined aggregation of the results. This thesis will solely look at File-level hotspots.

Knowledge Loss Knowledge loss in CodeScene is calculated by determining how much of the code was written by former contributors. A contributor is considered former when they have not contributed to the code in the last six months. This metric is based on the historical LoC contributed by each developer [11]. Additionally, modules where former contributors have contributed at least 50% of the code are flagged as experiencing Knowledge Loss.

Knowledge Distribution The knowledge metrics in CodeScene are based on the code contributions made by each developer [16]. CodeScene uses the extensive history of each file to compute these contributions. Because of that even if a developer rewrites a section of code, the original author still possesses knowledge in that area. This is acknowledged in CodeScene's metrics, and therefore the original author will have some knowledge retention accordingly.

Code familiarity and Knowledge islands are the two sub-metrics of knowledge distribution. Code Familiarity shows what percentage of the codebase the current active developers are familiar with, based on their experience with the files. Knowledge islands show a percentage of how much of the codebase is only known by one developer. Both these values can indicate knowledge risk, referring to gaps in knowledge or dependencies on individuals.

Main Author CodeScene identifies the author who has historically written the most LoC in a particular file or component which CodeScene calls the main author [11].

Author Churn Diagram The author churn diagrams created in CodeScene will be utilized to illustrate the effects of introducing and removing team members from a project. These diagrams visually represent the incurred expenses resulting from onboarding processes and reduced system familiarity [13].

The author churn diagram provides insight into several key aspects, including the number of active authors, their experience level months of experience, maximum possible experience within the team, onboarding costs, and qualitative team experience. The experience level is based on number of active months and divided into three categories: on-boarded (0-3 months), experienced (6-12 months), and veterans (+12 months) [13]. An example of an author churn diagram can be seen in Figure 2.1. As seen in the figure, the dark orange color in the bars represents the veteran, whereas the medium orange represents the experienced and light orange the onboarded.

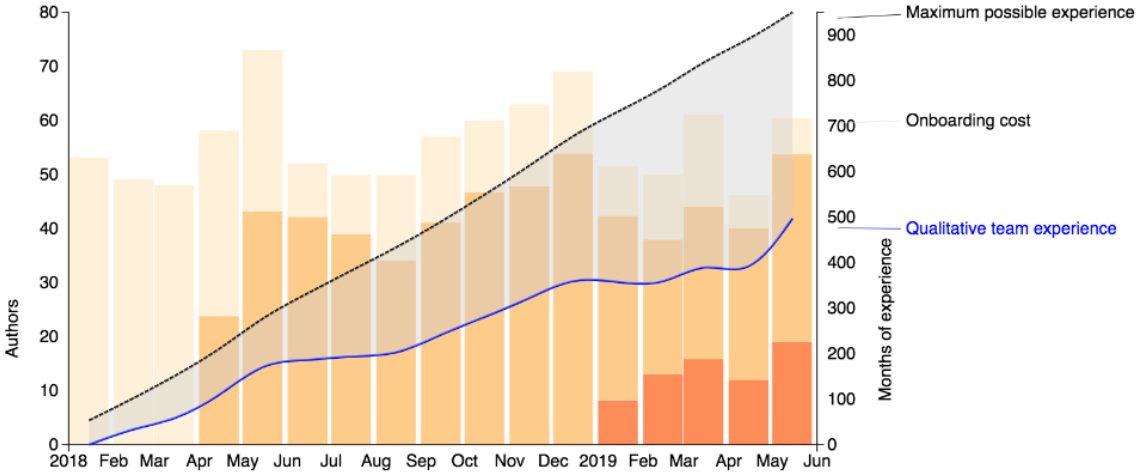


Figure 2.1: Example of the author churn trend diagram. Accessed from Codescene [13] 2024-04-15

It should be noted that the authors churn trend diagram presented in this thesis will be a modified version of the one found in CodeScene. Specifically, it will solely focus on displaying the experience levels (onboarding, experienced, veteran) of current developers in each project.

Issues Categorized As Defects Issues tagged as defects are classified and estimated in CodeScene using commit message patterns [14]. For the analysis made for this thesis, the ticked ID pattern matching GitHub's issue numbers is used to identify issues classed as defects.

Chapter 3
Case Study Design

This chapter presents the design of the case study, introducing the following components of this case study design: constructs, objectives and research questions, the criteria for case selection, data collection procedure, analysis procedure, and measures taken to ensure the validity of the findings. This thesis has followed the guidelines for conducting and reporting a case study research by Runeson et al. [72]. The decision to conduct a case study in this context is due to its capacity to provide a comprehensive and in-depth understanding of the studied Open Source Software (OSS) projects.

3.1 Constructs

This section provides a list of the key constructs that are used throughout this thesis. It starts by introducing the research on which these constructs are based on, followed by a list of the construct and their definition.

Contributor, Leaver, and Switcher

Nassif and Robillard, along with Izquierdo-Cortazar et al., offer insights into ways to categorize contributors within OSS projects. They define an 'active contributor' or 'active member' as a developer who consistently commits to a project's codebase [53, 43]. This thesis refers to a 'Contributor' as a developer who produced at least 1 commit during the last 6 months.

Furthermore, Nassif and Robillard identify a developer who ceases contributions to a project, indicated by a lack of commits, as a 'Leaver' [53]. Izquierdo-Cortazar, D., et al., call this a 'Non-active committer', which is described as someone who has not produced a commit to the project for one year [43].

This research defines it by describing a 'Leaver' as a contributor who has not made at least one commit to the project's codebase within the preceding six months, instead of one

year, to allow for a more responsive and accurate categorization of contributors' engagement level.

In this thesis, a third categorization is introduced to distinguish contributors who after the fork transition from one project to actively engage in the other project. This can be both from the original project to the fork, as well as from the forked project to the original project. Such a contributor is referred to as a 'Switcher'. This classification serves to acknowledge and analyze the patterns of contributors in the situation of a fork, to capture the transfer of contributors and their knowledge between projects.

Abandoned code

To define what constitutes abandoned code, research by Nassif and P.Robillard, as well as Izenquierdo-Cortazar et al., have proposed criteria for identifying such code. Nassif and Robillard regard a line of code as abandoned if written by a 'Leaver' [53]. Similarly, Izquierdo-Cortazar, D., et al., define a line of code written by a 'non-active committer' as an 'orphaned line' [43].

CodeScene's model for measuring knowledge and determining if code is abandoned differs. As stated in [16], CodeScene considers each file's history when determining if code is abandoned. CodeScene calculates the amount of code each contributor has written and whether the contributors are still active or not. Code is considered abandoned if it was written by a former developer. This thesis uses CodeScenes's model for determining "Abandoned Code" which takes into consideration the history of the code.

Knowledge Loss

In CodeScene, knowledge loss refers to code authored by contributors regarded as former contributors [16]. Former contributors can be manually configured in CodeScene. This thesis have regarded contributors who have not contributed within six months as former contributors, which corresponds to the default value in CodeScene. How the knowledge loss is calculated in CodeScene is described in section 2.5.In this thesis, knowledge loss will be examined in multiple ways, looking at both Knowledge Loss Scope and Knowledge Loss Duration.

3.1.1 List of Constructs

The following list introduces constructs that are used in this research:

- **Contributor**: This construct refers to an individual who has made at least one commit to the project's codebase within the preceding six months.
- **Leaver**: This construct refers to an individual who has not made at least one commit to the project's codebase within the preceding six months.
- **Switcher**: This construct refers to an individual who after the fork was a contributor in one project and then switched to become a contributor in the other project.

- **Abandoned Code**: This construct refers to CodeScenes's model for determining Abandoned Code, taking into account the code's history. This construct is used to measure knowledge loss.

- **Knowledge Loss**: This construct refers to the diminishing or disappearance of source code knowledge within an OSS project. The construct is a combination of *Knowledge Loss Duration* and *Knowledge Loss Scope*, which is calculated separately.

- **Knowledge Loss Scope**: This construct refers to the extent of the knowledge loss, i.e., the amount of orphaned or abandoned code in relation to the overall code base.

- **Knowledge Loss Duration**: This construct concerns the time it takes for a project to regain lost knowledge after a project fork, to same levels seen prior the fork.

- **Code Quality**: This construct refers to CodeScene's Code Health, an aggregated metric that has been validated for this purpose in previous research [77].

3.2 Objectives and Research Questions

The objective of this thesis is to explore the impact of forking on knowledge distribution and code quality in an OSS context. This thesis investigates two OSS projects and their corresponding forks in an exploratory case study. Aiming to investigate how knowledge distribution and code quality are affected in projects involved in a fork, and in doing so, gain new insights about knowledge loss and code quality under these circumstances.

To explore this, the following research questions are investigated.

RQ1 How can the knowledge loss resulting from the forking be characterized in terms of 1) Knowledge Loss Duration and 2) Knowledge Loss Scope?

RQ2 How is the knowledge loss from forking associated with changes in code quality?

RQ3 What is the association between forking, particularly in scenarios involving significant knowledge loss, and the frequency of bug reports in software modules subsequent to the forks?

RQ4 How did the OSS communities mitigate the knowledge loss from forking?

3.3 Selection Criteria

This section outlines the established criteria used in the various selection processes within this thesis. It introduces the criteria applied when selecting projects for examination, as well as the criteria for selecting hotspot files as well as primary contributors.

3.3.1 Case Selection

This section introduces the established criteria for selecting projects for this thesis. The chosen criteria were adapted from prior studies but tailored to the specific context of this research. For this thesis, two projects and their corresponding forks were selected as cases for investigation. These projects are described in detail in chapter 4.

Project Criteria

When selecting projects for evaluation, accessibility, community engagement, and project size were prioritized aspects. GitHub's prominent role as a hosting platform [20] was deemed a primary criterion (C1) for project selection, ensuring easy access to the chosen projects. Additionally, the presence of an active and engaged community was prioritized. Prior research has utilized a metric known as the popularity score [53], which mandates that the chosen project must have a minimum of 100 stars on GitHub. Therefore, the second criterion (C2) stipulates that the project must have a minimum of 100 stars on GitHub, ensuring the popularity and engagement of the chosen project(s).

Furthermore, the number of contributors was also a significant factor. Previous studies have advocated for a criterion of at least 50 contributors to mitigate the risk of knowledge loss inherent in smaller projects [53]. Thus, the third criterion (C3) mandates that a project must have a minimum of 50 contributors to be considered. Project size was also a crucial factor. Larger projects often demonstrate greater maturity and reduced reliance on individual developers [53]. Therefore, the fourth criterion (C4) stipulates that a project must exceed either 100 MB (medium) or 1 GB (large) in size to be considered. In line with this, to ensure substantial project size, the fifth criterion (C5) requires a minimum of 500 code files.

Considerations of historical context and the occurrence of forks were deemed essential to the selection process. Projects created before 2018 (C6) and forks happening before 2022 (C7) were considered. These criteria were chosen to ensure that the selected project(s) possessed substantial historical backgrounds and enough data points to be compared effectively. Additionally, requiring the original project to have at least one commit every six months for the two years leading up to the fork (C8) further ensures that there is sufficient data available for comparison.

The selection process for the project(s) in this case study also took into account the various reasons driving hard forking within OSS. Acknowledging the significance of diverse motivations behind a hard fork, additional criteria were established to ensure a diverse representation of projects, each exhibiting distinct reasons for initiating the fork. However, this criterion is considered more "nice-to-have" rather than one that must be fulfilled.

In summary, the project criteria (C) that each project must fulfill are as follows:

C1 The codebase must be hosted on GitHub.

C2 A GitHub repository with at least 100 stars.

C3 A minimum of 50 contributors.

C4 Larger than 100 MB (medium) or 1 GB (large).

C5 A minimum of 500 code files.

C6 The original project must be created four years prior to the current date.

C7 The fork must have happened two years prior to the current date.

C8 The original project must have had at least one commit every six months for the two years leading up to the fork.

C9 The chosen project must exhibit distinct motivations for initiating the fork.

When searching for potential projects to study 11 projects that were forked were found. All 11 projects were evaluated to see if they met the criteria. After evaluating the projects, there were two projects and their corresponding forked projects that met all criteria. The two projects and their corresponding forks that were selected were; Elasticsearch and OpenSearch, as well as ownCloud and Nextcloud. A detailed overview of how these chosen projects fulfill the criteria is provided in Table 3.1, while a comprehensive list of all identified projects can be found in Appendix B. Further elaboration on the chosen projects is presented in chapter 4.

Table 3.1: How the chosen projects meet the criteria. Data sourced from GitHub, accessible via their respective GitHub pages, as of 2024-02-27. *Based on data from the repositories core (ownCloud) and server (Nextcloud).

	Project 1		Project 2	
	Elasticsearch	OpenSearch	ownCloud	Nextcloud
Hosted on GitHub (C1)	Yes	Yes	Yes	Yes
GitHub Stars (C2)	66.8k	8.3k	8.2k*	25.1k*
Contributors (C3)	1908	264	536*	908*
Project Size (C4)	784 MB	554 MB	342 MB*	2.89 GB*
Minimum of 500 files (C5)	Yes	Yes	Yes	Yes
Original Start Year (C6)	2010	—	2010	—
Year of Fork (C7)	—	2018	—	2016
Commit every 6 months (C8)	Yes	Yes	Yes	Yes
Forking Motivation(C9)	License Change		Community	

Repository Criteria

In addition to the project criteria, the selection of projects with a polyrepo architecture required additional considerations specific to that type. The polyrepo architecture is characterized by using multiple repositories, as opposed to a single one, for the source node[36]. Therefore, when selecting projects with a polyrepo architecture, the repository criteria that each repository must fulfill are as follows:

C10 The repository in the original project must have been initialized at least 2 years prior to the fork date.

C11 The repository in the forked project must have been initialized no later than 6 months after the creation of the fork

C12 The repository must have at least one commit in every six-month period in the two years leading up to the fork.

C13 The repository must demonstrate functionality that is present in both the original and forked projects, indicating a shared core feature set.

For this thesis, one project and its corresponding fork utilized a polyrepo architecture: ownCloud and Nextcloud. Table 3.2 provides a detailed overview of the chosen repositories and how they fulfill the criteria. All data utilized in this table has been sourced from GitHub, accessible via their respective GitHub pages. For a full list of all repositories, see Appendix C.

Table 3.2: How the chosen repositories meet the set criteria. Data sourced from GitHub, accessible via their respective GitHub pages, as of 2024-02-27.

Repository		Criteron			
ownCloud	**Nextcloud**	C10	C11	C12	C13
Activity	Activity	2014-05-03	2016-07-31	Yes	Yes
Android	Android	2012-08-25	2016-06-06	Yes	Yes
Android-library	Android-library	2014-01-18	2016-06-06	Yes	Yes
Core	Server	2012-08-25	2016-06-02	Yes	Yes
files_antivirus	files_antivirus	2014-11-13	2016-09-02	Yes	Yes
files_pdfviewer	files_pdfviewer	2014-05-03	2016-06-20	Yes	Yes
files_texteditor	files_texteditor	2014-02-06	2016-06-21	Yes	Yes
firstrunwizard	firstrunwizard	2014-05-03	2016-06-12	Yes	Yes
notes	notes	2013-03-18	2016-10-19	Yes	Yes
apps	apps	2012-08-25	2016-06-21	Yes	Yes
updater	updater	2014-05-03	2016-06-12	Yes	Yes
tasks	tasks	2014-03-28	2016-10-20	Yes	Yes
templateeditor	templateeditor	2014-06-12	2016-06-21	Yes	Yes

Folder Criteria

In a single repository architecture, the content difference between projects after forking can be significant, i.e., folders might be removed or introduced. To facilitate effective comparison, the focus was placed on comparing folders that serve similar purposes and exist in both projects. Consequently, the decision was made to exclusively examine folders with equivalent logic that are present in both the original project and its fork. Thus, folders unique to either project were excluded from the analysis.

In this thesis, one project and its fork, Elasticsearch and OpenSearch, consisted of a single repository architecture. Thus, only folders with similar logic were chosen for the analysis.

3.3.2 Hotspot File Selection

This section introduces the criteria used when establishing Hotspot files. In the context of software projects, it is not uncommon to encounter files that have remained unchanged for

extended periods. When a project is forked, unnecessary files might be inherited, potentially introducing noise when conducting measurements. To ensure a comprehensive understanding of knowledge distribution and its impact on the cases under study, Hotspot files were also investigated. Hotspot files are those that have undergone significant changes over a specific period, as detailed in section 2.5. These files were identified using CodeScene.

For this thesis, the chosen timeframe spans from the beginning of each project until one year following the fork. Accordingly, the analysis concentrates on files that have undergone significant changes during this period. These files have been chosen according to the predetermined values given by CodeScene.

It was observed that ownCloud contained 7 hotspot files, whereas Nextcloud had four. Among these, 2 files were identified as hotspot files in both projects. For Elasticsearch and OpenSearch, Elasticsearch comprised 9 hotspot files, while OpenSearch had 7. They shared 1 hotspot file. For a complete list of all Hotspot files, see Appendix E.

3.3.3 Key Contributor Selection

The rationale for contributor selection is twofold. Firstly, it aims to identify the key contributors in each project. Secondly, it aims to identify potential interview candidates, which will be further discussed in subsection 3.4.3. This selection is driven by the understanding that various contributors may hold varying levels of expertise relevant to the projects. Consequently, the departure or transition of certain contributors may impact project outcomes to a greater extent than others. To determine key contributors within each project, the metric of primary file ownership was employed. Key contributors are identified as contributors who are the primary owners of one or more percent of the total files within a project. This number was chosen since it corresponds roughly with the Pareto Principle, i.e., that 80% of the development stems from approximately 20% of the contributors [34]. The result of this can be seen in chapter 5.

3.4 Data Collection Procedure

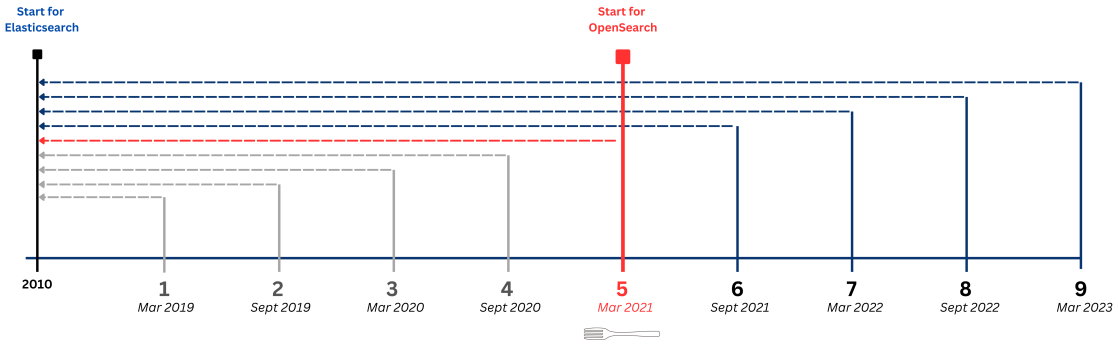
To answer the research questions, a combination of quantitative and qualitative data was gathered. It begins by introducing methods for gathering quantitative data followed by methods for gathering qualitative data.

3.4.1 Quantitative Data

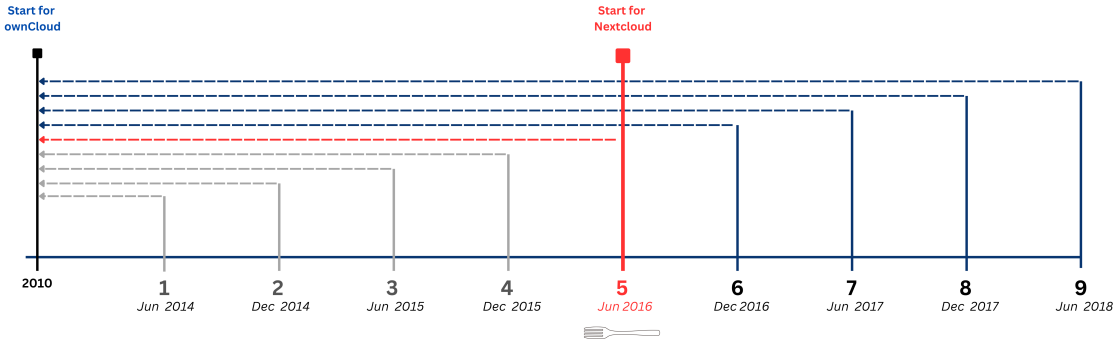
The quantitative data was sourced from GitHub repositories, and is categorized as third degree archival data, is pre-existing and not created for the purpose of a study, and in this case, this thesis [72]. Archival data is third degree data that can be sourced from for example a version management tools. Because of this, it should be acknowledged that it could lack certain desired elements in some cases, however not for this thesis. To decide what metrics to collect from the analyses the Goal Question Metric method was used, explained further in subsection 3.4.2.

The procedure for collecting quantitative data involved three primary stages: collection, preparation, and sorting. Initially, repositories relevant to each project were forked on GitHub, creating private copies.

The preparation stage included organizing the data within defined time periods, termed 'phases'. For this thesis, the time span was divided into nine phases spanning four years, commencing two years prior to the fork and two years thereafter. These phases aimed to capture the evolution of each project during these four years. Specifically, five phases were taken before the fork, referred to as 'baseline' phases, while four phases for each project were taken after the fork, referred to as 'subject' phases. These phases occurred biannually, where each phase encompasses data from the start of the original project up to its respective time point. A visual representation of each phase can be seen in Figure 3.1a for Elasticsearch and OpenSearch, and Figure 3.1b for ownCloud and Nextcloud. Each phase's temporal boundaries were determined by identifying the last commit on the original GitHub page of each repository, distinguished by its unique commit ID. In phases where no commit precisely matched the phase's end date, the closest commit preceding that date was selected.



(a) Elasticsearch and OpenSearch



(b) ownCloud and Nextcloud

Figure 3.1: Figures over the phases for (a) Elasticsearch and OpenSearch and (b) ownCloud and Nextcloud. The number refers to each phase.

Following this, the data was organized into these phases during the sorting stage. This

process involved creating a branch for each phase using its unique commit ID. Each branch included all history up to that specific point in time.

The following list outlines each step of the quantitative data collection process:

1. **Collection**: Generate a private repository copy on GitHub and clone it to the private workspace.

2. **Preparation**: Identify a unique commit for each analysis phase.

3. **Sorting**: Establish a branch for each analysis phase using the unique commit ID.

Throughout this process, nine branches were created for each repository within the original project, while four analyses were conducted for each repository for the forked projects. All data was sourced from the respective projects' GitHub pages, and a complete list of all repositories and commits used for data collection is provided in Appendix C.

3.4.2 Goal Question Metric

To decide what metrics to collect from the analyses, the Goal Question Metric method (GQM) was applied [72]. The collected metrics were based on their relevance to answering the research questions, which were derived from this thesis's aims and objectives outlined in section 1.1. The chosen metrics, categorized by research questions (see section 3.2), are presented in Table 3.3.

Table 3.3: Chosen metrics based on the GQM

Research Question	Sub-questions	Metric in CodeScene
RQ1	How high is the knowledge loss?	Knowledge Loss
		Knowledge Loss (For Hotspot files)
	How high is the code familiarity?	Code Familiarity
RQ2	What is the level of Code Quality in each subject after the fork?	Code Health
		Code Health (For Hotspot files)
RQ3	What is the bug frequency?	Issues classified as defects
		Issues classified as defects in Hotspot files
RQ4	Is the knowledge loss reduced over time?	Author Churn Diagram
		knowledge loss

3.4.3 Qualitative Data

This section describes the method employed to collect the qualitative data. To bridge potential gaps in the quantitative data, additional qualitative data was collected through inter-

views, providing first-degree data acquired in real-time through direct communication with respondents.

Semi-structured interviews were chosen as the primary method for gathering qualitative data, as it is a common method for this purpose [1]. Semi-structured interviews combine asking predetermined open-ended questions with the possibility for the interviewer to ask follow-up questions based on answers from the participants [1]. The interview followed a structured approach based on the Funnel Technique, comprising three distinct phases aimed at minimizing participant bias. Initially, an introductory phase to outline the objectives and context of the interview. Subsequently, a phase encompassing general inquiries related to open source was conducted. Finally, a more focused phase addresses the specific research questions explored in this thesis. This structured approach was adopted in alignment with the guidelines outlined by Runeson et al. [72]. The interview questions used in this thesis are detailed in Appendix A. The participants for the interviews were selected using Purposive Sampling, a method that involves intentionally selecting participants based on specific criteria, e.g., knowledge, experiences [30]. Hence, key contributors in all projects are considered potential interviewees. Details regarding how key contributors were selected is provided in subsection 3.3.3.

Throughout this process, all key contributors from the studied projects were invited to partake. Ultimately, only two interviews were conducted with key developers from the same project: one via Google Meet and one via email.

3.5 Analysis Procedure

The analysis procedure for this thesis adopts a longitudinal approach to examine the projects and their respective forks over time and was conducted using CodeScene. The process of the analysis unfolds through a structured sequence of four distinct steps summarized in a list at the end of this section.

The process begins with the initialization and configuration of project instances within the CodeScene environment. Each project instance included repositories relevant to the specific phase under examination. Before the analyses were run, all configuration settings were adjusted. The configuration used during the analysis can be found in Appendix D.

Each project underwent two analyses, one initial analysis for CodeScene to identify potential former contributors and one including former contributors in the analysis. Throughout this process, a total of nine analyses were conducted in CodeScene for each original project, while four analyses were conducted for each of the forks. From the analysis results, the metrics in CodeScene, specified in Table 3.3, were collected.

For Elasticsearch and OpenSearch, each project consisted of a single branch. For ownCloud and Nextcloud, due to their polyrepo architecture, each project consisted of multiple branches within a unified CodeScene project. The analyses covered nine projects each for the original projects Elasticsearch and ownCloud, and four projects each for the forks OpenSearch and Nextcloud. From the analyses, all the metrics listed in Table 3.3 were collected for each project's analysis result.

The following list summarises the analysis steps:

1. Initialize and configure projects in CodeScene.

2. Configure former developer by running first analysis of projects in CodeScene.
3. Run second analysis in CodeScene with complete configuration including former contributors.
4. Collect the metrics using the GQM.

3.6 Truck Factor

In addition to the analysis, the analysis data created using CodeScene was used to calculate the truck factor (TF) of the baseline projects. The truck factor is a value that indicates what danger it would be for a project to lose key developers [48]. It refers to the minimum number of core developers that have to be "hit by a truck" before the project is incapacitated [79]. A lower TF suggests that a project would be more impacted if a key developer left, and a high TF indicates that a project can handle losing developers better because of better knowledge distribution. The algorithm used to calculate the TF was the ALGO1, created by Andreas Karlsson [48], provided on GitHub by CodeScene [17]. This algorithm is considered state-of-the-art and has shown better results compared to previous algorithms found in research.

3.7 Validity Procedure

The validity procedure employed in this thesis adheres to the classification scheme used by Runeson et al. [72] and Yin [84]. The approach focused on four dimensions of validity: construct validity, reliability, internal validity, and external validity. Each dimension was analyzed, and specific countermeasures were implemented to address potential threats. It should be acknowledged that the third degree data has been produced for other purposes than this thesis, e.g., version control management of the projects. Thus, there could be uncertainty surrounding the completeness and validity of the data [72].

To ensure construct validity, data triangulation was employed. This involved collecting the same type of data from GitHub on multiple occasions to validate findings. Additionally, a thorough description of the chain of evidence, including the data collection and analysis methods, along with detailed documentation in the appendices, was provided to enhance transparency.

Reliability and internal validity were enhanced by involving multiple researchers in the data collection and analysis process. This approach helps mitigate biases and increases the consistency and dependability of findings. Moreover, criteria and construct were precisely defined and consistently applied across all research stages, minimizing the likelihood of interpretation errors.

External validity was addressed by employing a sampling technique and criteria that can be generalized and applied to similar contexts and projects beyond those in this research. Efforts were also made to provide detailed descriptions of all stages in the research, enhancing transparency, enabling reproducibility of the study's findings, and thereby strengthening reliability.

Chapter 4
Cases under Study

This chapter serves as an introduction to the cases under study for this thesis. Each section provides an overview of the projects encompassed within each case. This includes a detailed examination of each project, highlighting key descriptive elements, followed by a motivation and timeline of the events that precipitated the fork.

4.1 Case A: Elasticsearch and OpenSearch

This section introduces the first case under study: Elasticsearch and OpenSearch. It starts with providing a background of each project, followed by a description of the motivation behind the fork.

4.1.1 Elasticsearch

Elasticsearch is a distributed search and analytic engine, operating according to the REST principles [21]. It is designed to deliver both fast and relevant search results for large-scale operation workload [22].

The initial version of Elasticsearch was launched by Shay Banon in February 2010 [25], with its codebase licensed under the permissive Apache License version 2 (ALv2) [23]. Two years later, in 2012, Elastic NV was established to offer commercial services and products related to Elasticsearch and its associated software. Later, Elastic NV was re-branded to Elastic [18]. In 2021, Elasticsearch underwent a significant change in its licensing terms, moving from ALv2 to a dual licensing model comprising the Server Side Public License (SPPL) version 1 and the Elastic License 2.0 (ELv2) [23]. Neither of these licenses aligns with the Open Source Initiative or is included in the Free Software Foundation's list of free software licenses. Consequently, this alteration made Elasticsearch no longer available under an open source license [27]. For more details about licenses, see section 2.3. According to Elastic's website, this transition aimed to provide users with greater flexibility in selecting the license

that best suits their requirements [23]. Despite Elasticsearch not being licensed under Open Source Software (OSS) licenses, Elastic wanted to maintain and uphold the values of open source, including openness, community, and teamwork for the project. Elasticsearch's community remains active, and users can still contribute to its development. [23]

Today, Elasticsearch is one of many services provided by Elastic. As of the third quarter of Fiscal 2024, Elastic, the company behind Elasticsearch, reported that over 50% of the Fortune 500 companies were utilizing their services [28], achieving an annual revenue of $1069 million USD in 2023 [27].

In terms of Elasticsearch's descriptive assets gathered from the CodeScene analysis, Figure 4.1 illustrates the participation of active authors in the project over the years in blue, including post-fork. Initially, there was a consistent increase in active contributors until mid-2020, after which a decline set in, returning to levels reminiscent of those seen in mid-2014. In terms of project growth, Elasticsearch has experienced steady growth, averaging around 7500 Lines of Code (LoC) yearly, which can be seen in Figure 4.2.

4.1.2 OpenSearch

The company OpenSearch provides an open source search and analytics suite designed to provide users with a set of tools for search and analysis tasks. The suite includes various components, among them the search engine, OpenSearch, which is a fork from Elasticsearch. OpenSearch was introduced and launched in 2021 as part of this suite [61]. The first commits in the Github OpenSearch repository were on March 13, 2021, which in when the fork derived from Elasticsearch was made [62]. Throughout this thesis, the term "OpenSearch" will specifically refer to the search engine, while the corporate entity named OpenSearch will be referred to as the OpenSearch Company.

OpenSearch is hosted on GitHub and operates under the permissive Apache License, Version 2.0 (ALv2). The project is actively maintained and developed by their community, which includes a network of partners [61]. Notably, among these partners is Amazon Web Services (AWS) [63]. Despite the different partnerships, the projects remain open to contributions from individuals beyond its community and partner network [61].

In terms of OpenSearch's descriptive assets, the number of contributors over time is shown in orange in Figure 4.1. As seen in the figure the number of contributors increases over time, except at the end of 2022, where a decline started. Regarding project growth, OpenSearch has a slight increase in size over the time of this analysis which can be seen inFigure 4.2.

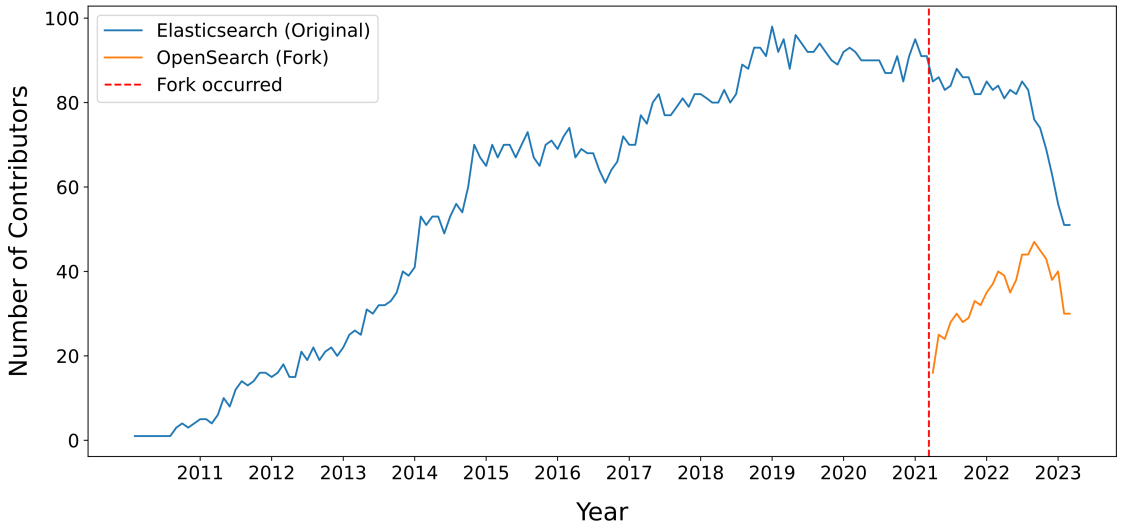


Figure 4.1: Active authors in Elasticsearch and OpenSearch. *Please note that for Elasticsearch, this graph is based on only the files that passed the selection criteria.*

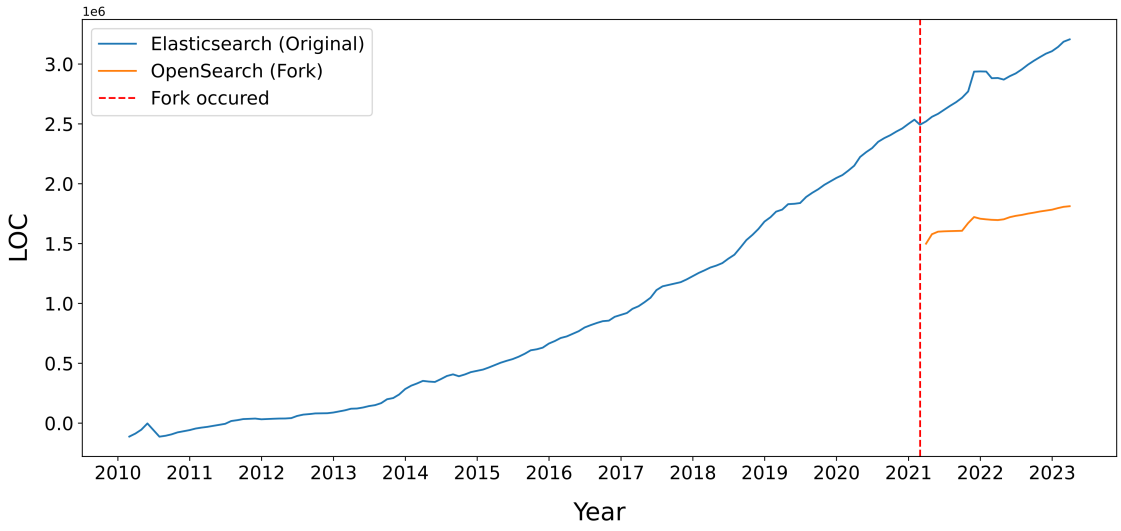


Figure 4.2: Project growth in Elasticsearch and OpenSearch. *Please note that this graph is based on only the files that passed the selection criteria.*

4.1.3 Motivation Behind the Fork

Drawing from various sources, it becomes evident that the primary motive for the fork leading to the development of OpenSearch stems from a shift in licensing. As articulated in a blog post by developers at AWS, who are also part of the OpenSearch contributor team,

> *"Last week, Elastic announced they will change their software licensing strategy, and will not release new versions of Elasticsearch [..] under the Apache License, Version 2.0*

> *(ALv2). Instead, new versions of the software will be offered under the Elastic License (which limits how it can be used) or the Server Side Public License (which has requirements that make it unacceptable to many in the open-source community). This means that Elasticsearch [..] will no longer be open-source software. In order to ensure open source versions of both packages remain available and well supported, including in our own offerings, we are announcing today that AWS will step up to create and maintain a ALv2-licensed fork of open source Elasticsearch and Kibana."* [51]

To understand the rationale behind this decision, it is crucial to understand the event and decisions that led to the creation of OpenSearch. These have been visually represented in a timeline presented in Figure 4.3, which highlights key events.

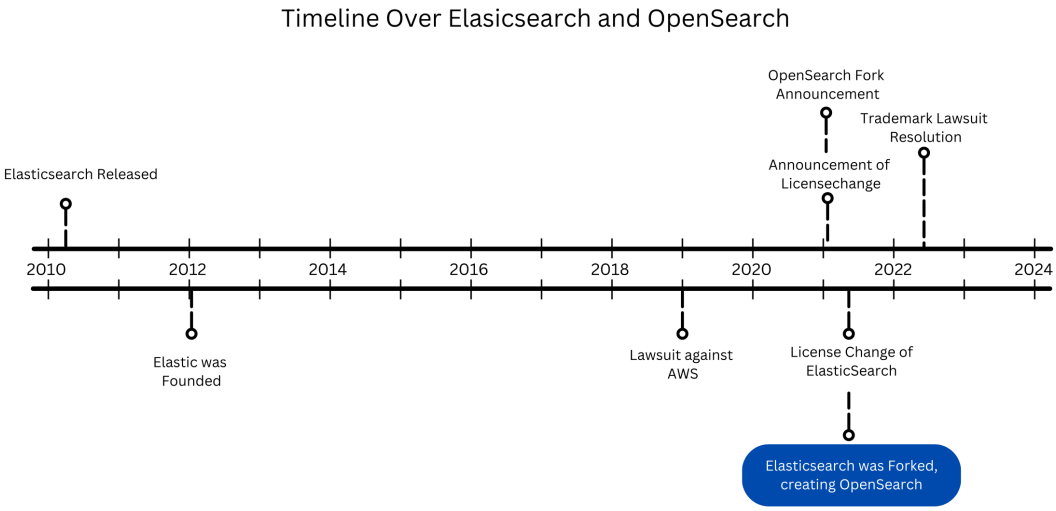


Figure 4.3: Timeline over key events in the creation of OpenSearch.

The timeline begins with the establishment of Elasticsearch, followed by the formation of Elastic later on. Over time, Elastic grew increasingly frustrated due to the perceived misuse of Elasticsearch [3]. This frustration culminated in legal disputes, including lawsuits [2, 3], one of which was filed by Elastic against AWS in 2019, citing trademark infringement and unfair competition [3]. The origins of this lawsuit can be traced back to events in 2015 when Amazon introduced a new service called Amazon Elasticsearch Service. In a January 2021 blog post, Shay Bannon directly addressed this issue, citing AWS and Amazon Elasticsearch Service as motivating factors behind the licensing change [3]. It has also been described as, *"This was a response to Amazon's non-collaborative behavior and misuse of our trademark. [..]"* [26].

In February 2021, Elastic introduced a new version of Elasticsearch under the new dual-licensing model [26]. AWS responded to this change by announcing, in March 2021, the creation of OpenSearch [74]. They motivated the fork by stating that it was aimed at preserving the software's open-source nature and avoiding vendor lock-in, as Elasticsearch would no longer be open source [74]. This marked the beginning of OpenSearch. Notably, in February 2022, it was announced that Elastic and Amazon had reached an agreement on the trademark lawsuit [3].

4.2 Case B: ownCloud and Nextcloud

This section introduces the second case under study: ownCloud and Nextcloud. It begins with providing a background of each project, followed by a description of the motivation behind the fork and timeline.

4.2.1 ownCloud

ownCloud, established in 2010, operates as a cloud storage platform combining both open-source and proprietary components [64]. Founded by Frank Karlitschek, a dedicated open-source contributor and privacy advocate [45], ownCloud emerged as an open-source alternative to proprietary services like Dropbox and Google Drive. The platform was designed to empower users with complete control over their data through a self-hosted infrastructure. Its popularity quickly grew among businesses and individuals, leading to the establishment of ownCloud Inc. on December 13, 2011, a company founded by Frank Karlitscheck, Markus Rex and Holger Dyroff. Today, it is managed by ownCloud GmbH in Nuremberg, Germany, and led by CEO Tobias Gerlinger and co-founder and COO Holger Dyroff [64].

ownCloud is distributed under the GNU Affero General Public License version 3 (AGPLv3), with its Enterprise Subscriptions governed by the ownCloud Commercial License [66]. The source code is distributed and maintained through GitHub.

In terms of ownCloud's descriptive assets, Figure 4.4 illustrates the number of contributors over time in blue. ownCloud had an increase in contributors until the end of 2014 when it started decreasing, continuing to the end of this analysis. Notably, just before the fork, the number of active contributors became notably lower.

Regarding ownCloud's project growth, as depicted in blue in Figure 4.5, significant changes are observed over the years. At the beginning of 2012, there was a drastic increase in LoC, followed by oscillations until mid-2014, when a rapid decrease occurred. Subsequently, there is a steady increase in LoC, with a notable exception occurring just before the fork, characterized by another substantial upsurge.

4.2.2 Nextcloud

Nextcloud is an open-source file synchronization and sharing solution that originated from the ownCloud project. Frank Karlitschek, who also founded ownCloud, introduced Nextcloud in June 2016 via a blog post [54]. Simultaneously, Nextcloud GmbH was established, with Karlitschek serving as its Chief Executive Officer [46, 55].

Nextcloud hosts its projects on GitHub, with all software licensed under the GNU Affero General Public License version 3 (AGPLv3) [56, 57]. By the end of 2017, Nextcloud had expanded its team to 35 employees [54].

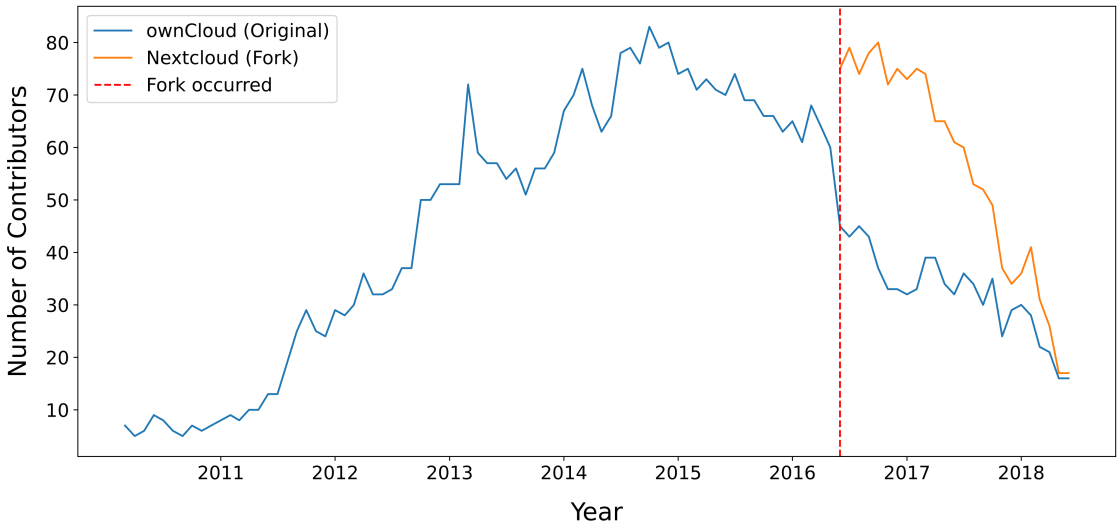


Figure 4.4: Active authors across all files in ownCloud and Nextcloud.

In terms of Nextcloud's descriptive assets, the orange line inFigure 4.4 displays the number of active contributors in Nextcloud. The figure reveals that the project has a notable amount of active contributors at the start. However, less than a year after its creation, there is a consistent decline in active contributors throughout the entire analysis period.

Regarding Nextcloud's project growth, displayed in orange inFigure 4.5, the figure reveals that it has significantly changed over the years. Initially, there was a steady increase in lines of code. However, in mid-2017, a significant decrease is observed. This is followed by a steady, but slow, increase, albeit not reaching the same level as before the decline.

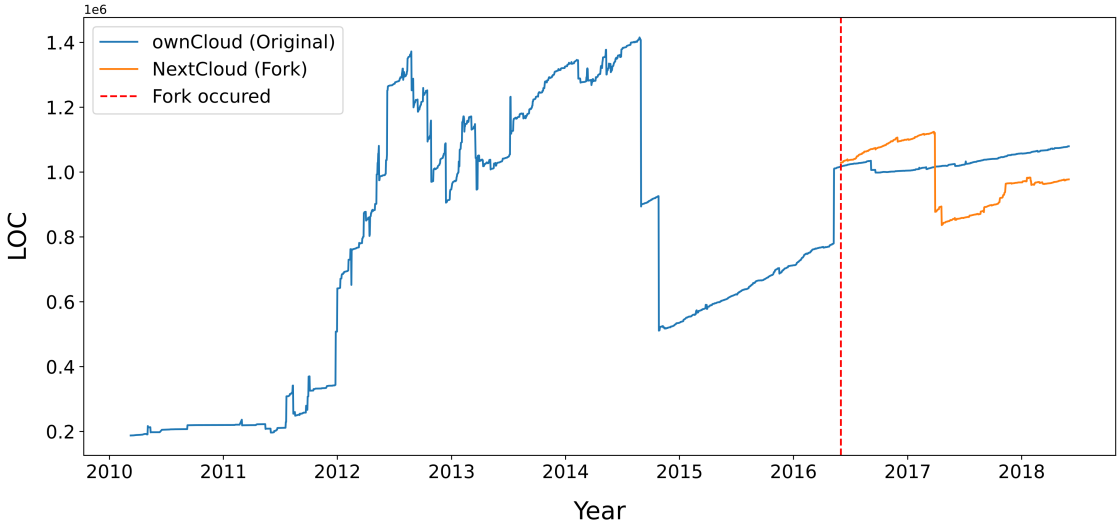


Figure 4.5: Project growth in ownCloud and Nextcloud

4.2.3 Motivation Behind the Fork

The history of ownCloud and Nextcloud, depicted in the timeline shown in Figure 4.6, begins in 2010 when Frank Karlitschek, Tobias Gerlinger and Holger Dyroff founded ownCloud. Subsequently, ownCloud Inc. was established in 2011, to provide support services, and facilitate commercial partnerships [64].

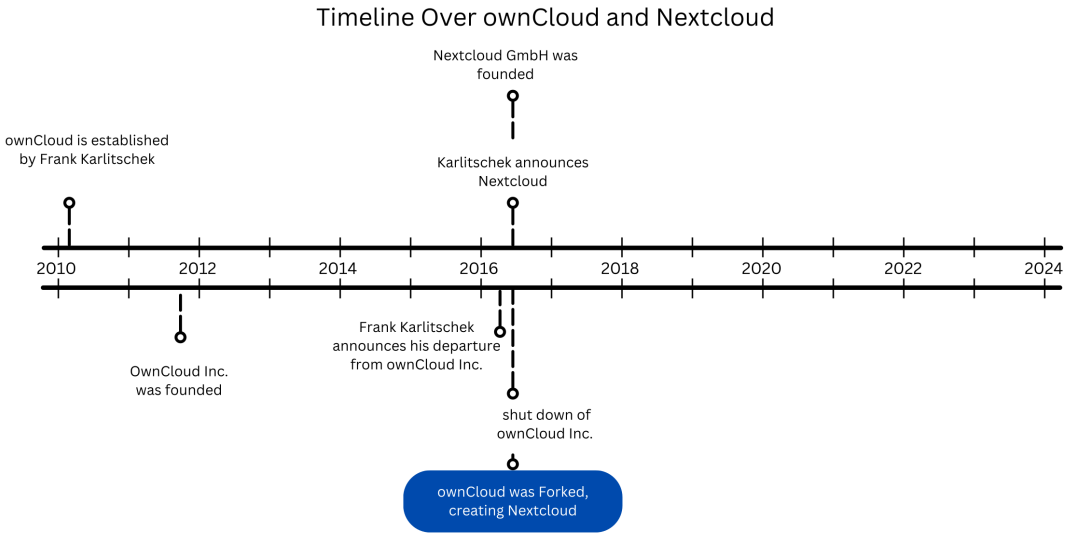


Figure 4.6: Timeline over key events in ownCloud and Nextclouds history.

On April 27, 2016, Karlitschek announced his departure from ownCloud, Inc. on his personal blog [46]. He also mentioned that his journey with ownCloud had not come to an end. Additionally, he explained that the community and the company behind ownCloud had worked well together to release version 9.0 of ownCloud. However, he noted that there were experiences that differed from the previously successful collaboration, which put them in a complicated position, although he did not elaborate on this. He went on to discuss some moral questions he had considered:

> *"Who owns the community? Who owns ownCloud itself? And what matters more, short-term money or long-term responsibility and growth? Is ownCloud just another company, or do we also have to answer to the hundreds of volunteers who contribute and make it what it is today?"* [47].

On June 2, 2016, Karlitschek announced the forking of ownCloud to create the new project called Nextcloud in another blog post [46], together with a team of core developers from ownCloud. In this post, he explains that *"ownCloud own one of the biggest and most important open source projects"*. He also informed that he, along with*"most of the core technical people"* from ownCloud, would create Nextcloud by forking the ownCloud project. Nextcloud GmbH was founded on the same day. In response to this announcement, ownCloud wrote a news post stating that Karlitschek's announcement both surprised and disappointed them [67]. They addressed the criticism regarding the community and stated their intention to strengthen it by forming an ownCloud Foundation, consisting of seven members who are

community users from the GitHub community and ownCloud representatives. In the same post, they announced the closure of ownCloud, Inc. due to their credit being canceled by their main lenders in the US, and terminated 8 employees' employment agreements. However, they clarified that this did not directly affect ownCloud GmbH [67].

Based on these posts, it indicates that the primary motivation behind the fork is due to governance disputes, explained in section 2.2.

Chapter 5
Results

This chapter presents the research findings for this thesis, and is structured around each case study. Note that this chapter presents only the objective results; discussions surrounding the results are in chapter 6. It begins with results for Case A: Elasticsearch and OpenSearch, followed by Case B: ownCloud and Nextcloud. Each case covers results related to contributors, knowledge loss, code familiarity, and code quality, concluding with code defects. Additionally, each subject (except contributors) also includes a presentation on hotspot files.

5.1 Case A: Elasticsearch and OpenSearch

This section presents the results for Case A: Elasticsearch and OpenSearch. Each segment starts with a summary of key findings, followed by a detailed examination of the subject matter.

5.1.1 Contributors

This subsection addresses results on Key Contributors, Contributor Turnover Rate and Author Churn Trend.

Summary of Section: Contributors

Elasticsearch initially had 30 key contributors, one of whom switched to OpenSearch within 6 months. OpenSearch had only two key contributors, both of whom remained. Post-fork, OpenSearch experienced a higher turnover rate, gaining an average of 8.75 contributors every six months, while Elasticsearch lost on average 4. The author churn trend revealed that the maturity levels of the Elasticsearch project remained relatively stable post-fork, whereas they significantly declined for OpenSearch.

Key Contributors

The findings regarding key contributors selection for Case A, described in subsection 3.3.3, are presented in Table 5.1. The analysis uncovered that Elasticsearch had 30 key contributors, constituting 3,61% of all developers, all of them had been active during the analysis period of this thesis. OpenSearch had 24, approximately 2.40% of all their developers. However, 22 of these contributors made their last commit in Elasticsearch before the fork and were never active in OpenSearch. Therefore, they are excluded as key contributors for OpenSearch in this thesis, leaving OpenSearch with only 2 key contributors.

Table 5.1: Data used to determine key contributors. 'Nbr of File Owners' represents all contributors that own a file in the project. Furthermore, 'Nbr of File Owners ≥ 1%' represent the amount of contributors that owns ≥ 1% of the files in the project, also referred to as 'key contributors'. Lastly, the percentage represents the proportion of file owners relative to the total number of contributors. Note that file owners can be both former and current contributors.

Project	Nbr of Files	Nbr of File Owners	Nbr of File Owners ≥ 1%
Elasticsearch (Original)	8833	190 (22,92%)	30 (3,61%)
OpenSearch (Fork)	8486	223 (22,4%)	24 (2,40%)

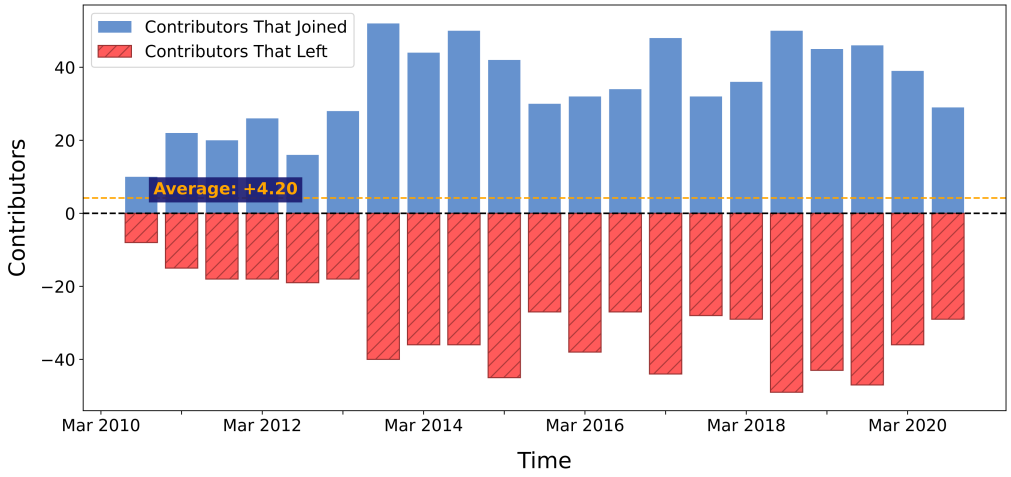
Contributor Turnover Rate

The contributor turnover rate for Case A, Elasticsearch and OpenSearch, is illustrated in Figure 5.1.

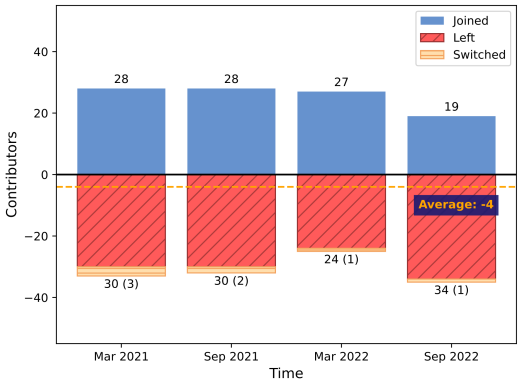
Elasticsearch before the fork, referred to as 'baseline', had a turnover rate of +4.20 new contributors every six months, as shown in Figure 5.1a. Following the fork, Figure 5.1b demonstrate that Elasticsearch (original) saw a decrease in contributors with a turnover rate of -4 developers per six months, whereas Figure 5.1c shows that OpenSearch (Fork) witnessed an increase in contributors with a turnover rate of +8.75 developers per six months. The turnover rate was overall higher in OpenSearch (Fork) compared to the baseline and Elasticsearch (Original).

Regarding the turnover of key contributors, the analysis uncovered that during the first phase after the fork, one key contributor departed from Elasticsearch (Original). At the same time, there was no turnover of key contributors in OpenSearch (Fork).

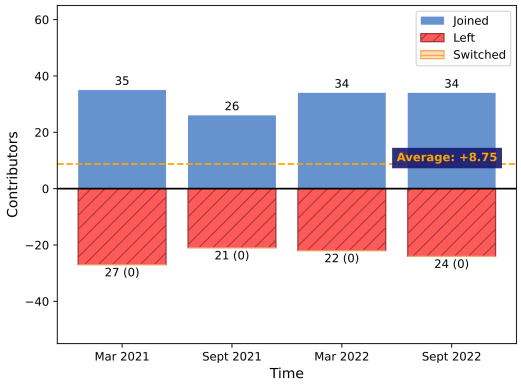
The analysis also compared turnover rates between the projects, identifying 7 contributors involved in the development of both projects post-fork. Among them, all 7 contributors transitioned to exclusively develop in OpenSearch within the initial six months following the fork.



(a) "baseline", before the fork.



(b) Elasticsearch (original)



(c) OpenSearch (fork)

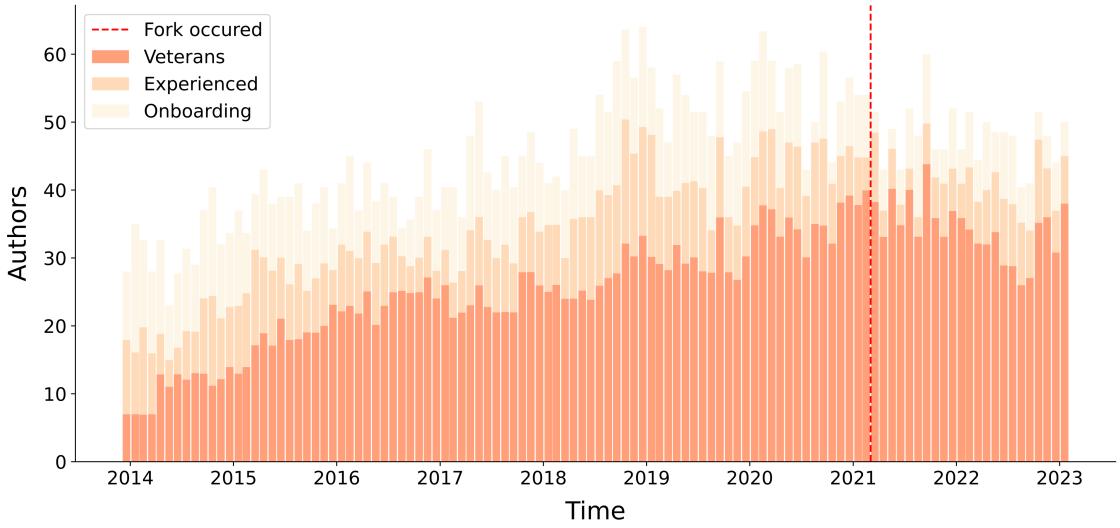
Figure 5.1: Turnover rate for (a) Elasticsearch before the fork, i.e., "Baseline", (b) Elasticsearch (original project) and (c) OpenSearch (fork) after the fork. Each bar represents six months.

Author Churn Trend

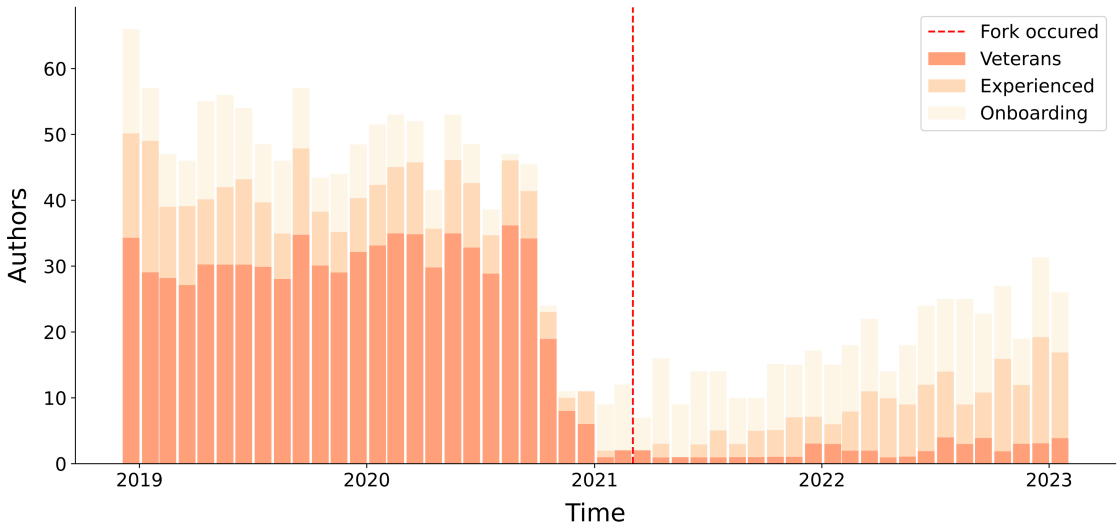
The author churn trend for Elasticsearch is illustrated in Figure 5.2a. Prior to the fork, there was a steady increase in veteran, experienced and onboarding contributors. Post-fork, the number of veterans remained somewhat stable throughout the analysis, with one occasional drop in late 2022. The number of experienced contributors witnessed a decline in the months following the fork but eventually stabilized at levels comparable to those observed before the fork by the end of 2021. Furthermore, the number of onboardings decreased after the fork, with a slight increase observed in mid-2022.

In Figure 5.2b, the author churn trend for OpenSearch is presented. The trend is depicted by a red dotted line presenting the post-fork period. This trend is then compared with the churn trend for the same code in baseline, which represents the period before the fork. The figure reveal that the number of veteran contributors significantly decreased after the fork, with occasional small increases. The number of experienced contributors steadily increase

over time post-fork, although started at lower level compared to the baseline. Furthermore, the number of onboarding contributors increased compare to levels found in baseline, but remained relatively constant throughout all periods post-fork.



(a) Elasticsearch



(b) OpenSearch

Figure 5.2: Author Churn Rate for each project in case A: (a) Elasticsearch and (b) OpenSearch. Each bar represents a month, with the red dotted line indicating the date of the fork.

5.1.2 Knowledge Loss and Code Familiarity

This section introduces the findings regarding knowledge loss across both project and in hotspot files, alongside an assessment of code familiarity.

Summary of Section: Knowledge Loss and Code Familiarity

OpenSearch (Fork) experienced greater knowledge loss and lower code familiarity post-fork compared to Elasticsearch (Original). However, both projects did experience higher knowledge loss compared to baseline. Knowledge loss was notably higher in all hotspot files for OpenSearch.

Figure 5.3 illustrates the Average Knowledge Loss for both projects in Case A throughout the analysis time period. The figure shows a significant increase in knowledge loss for the forked project (OpenSearch) compared to both the baseline and the original project (Elasticsearch). Within two years, the average knowledge loss in the forked project (OpenSearch) had not yet stabilized to the levels observed in the baseline. For the original project (Elasticsearch) there was a slight increase in knowledge loss in the second to last phase, followed by a decrease in the last phase.

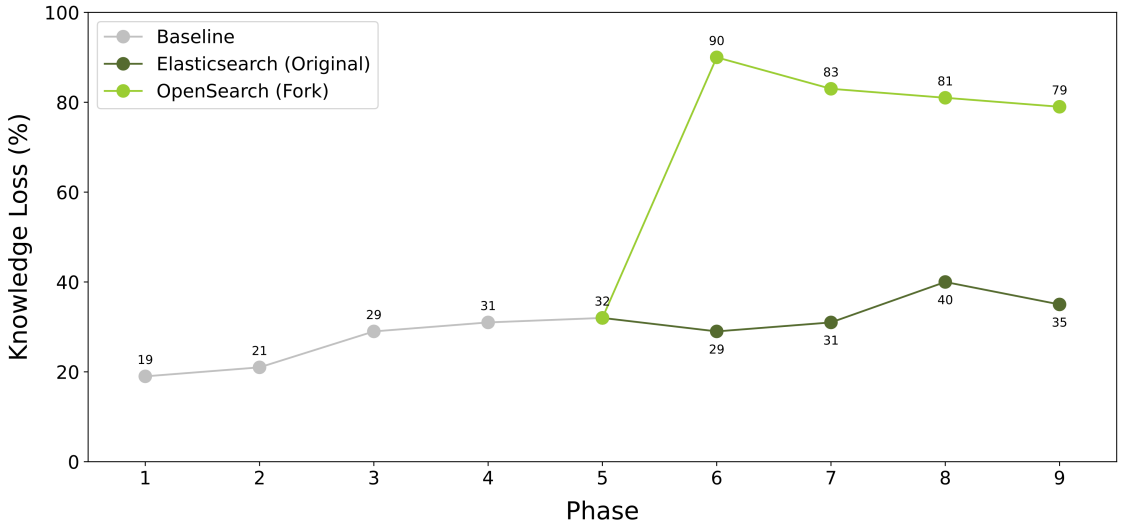


Figure 5.3: Average Knowledge Loss in Elasticsearch and OpenSearch. Baseline refers to the original project before the fork, i.e., Elasticsearch.

A comparison of the average knowledge loss for each hotspot files, one year post-fork, is illustrated in Figure 5.4. The figure consist of 17 hotspot files, where File 1 is a hotspot file in both projects, wile Files 2 through 10 are hotspot files in Elasticsearch, and Files 11 through 17, are hotspot files in OpenSearch. Overall, OpenSearch exhibits higher average knowledge loss across all hotspot files compared to Elasticsearch. Particularly, among the hotspot files found in OpenSearch, there is a consistent trend of elevated knowledge loss.

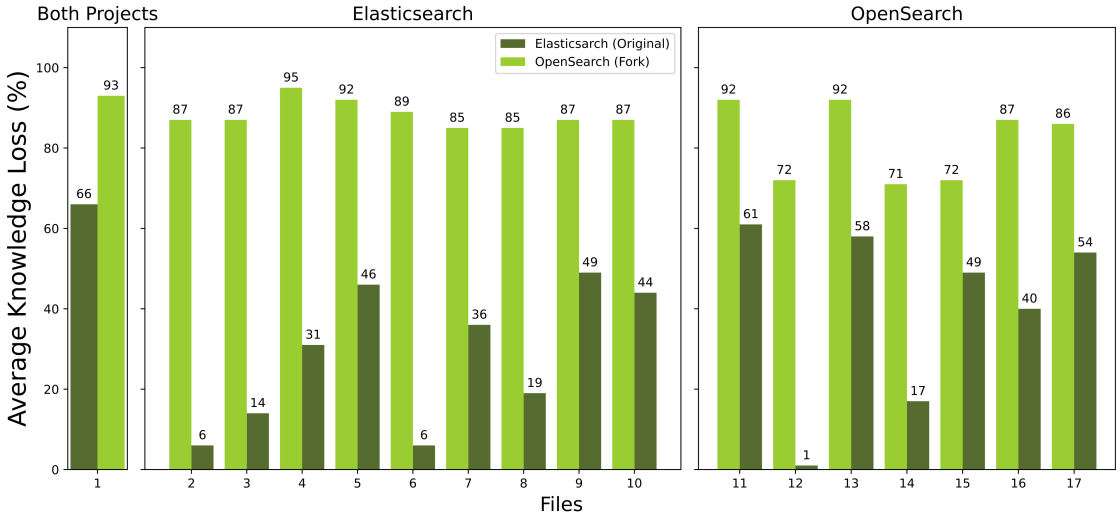


Figure 5.4: Average knowledge loss after the fork for Case A: Elasticsearch and OpenSearch.

Turning to code familiarity, Figure 5.5 delineates the changes in code familiarity for both projects throughout all phases. It reveal a decline in code familiarity for both Elasticsearch and OpenSearch post-fork, with OpenSearch experiencing the most significant decrease, and much lower code familiarity than Elasticsearch. For OpenSearch, after the initial decrease, an increase followed that continued until the last phase. The small decline in Elasticsearch persisted over almost the whole analysis period, with only minor fluctuations.

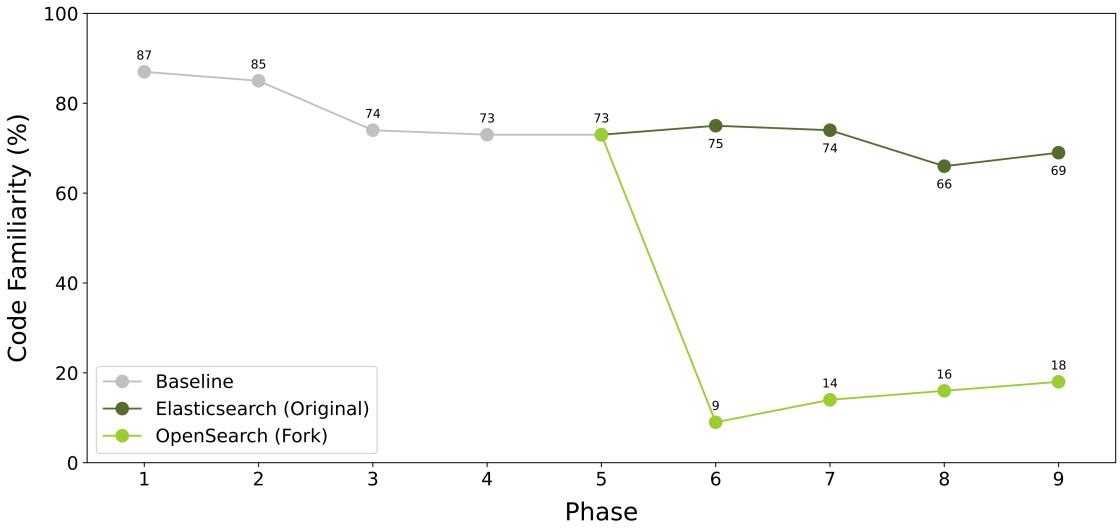


Figure 5.5: Average Code Familiarity for Elasticsearch and OpenSearch. Baseline refers to the original project before the fork, i.e., Elasticsearch.

5.1.3 Code Quality

This section presents the results related to code quality, it begins with the overall project followed by findings related to each project's hotspot file.

Summary of Section: Code Quality

There were consistent trends in code quality for both projects with a slight decline for both projects. Elasticsearch increased in size post-fork, while OpenSearch decreased. Hotspot file analysis shows a decline in Code Quality in Elasticsearch and varying trends in OpenSearch.

The analysis of code quality and project size, as depicted in Figure 5.6, reveals insights into the progression of both metrics across all phases. The figure illustrates a consistent trend in code quality for both projects throughout the phases, with a slight decline observed post-fork, notably from phases 6 to 7. However, from phases 7 to 9, the code quality remains relatively stable. In terms of project size, Elasticsearch demonstrates a increase, post-fork, that remained consistent for the subsequent phases, whereas OpenSearch (Fork) experiences a significant decrease compared to the baseline.

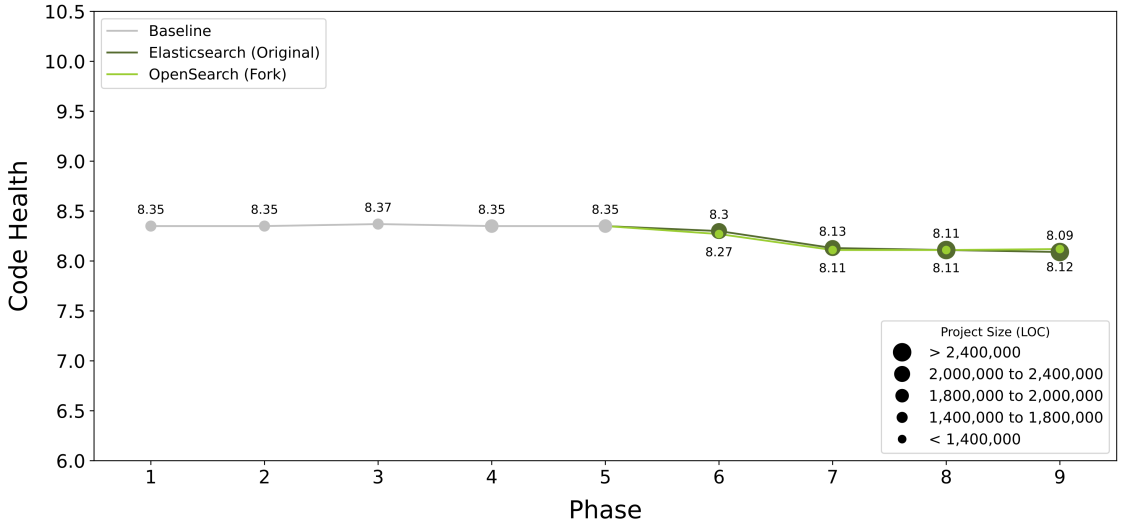


Figure 5.6: Code health and project size for Elasticsearch and OpenSearch. Project size is measured in lines of code (LoC). Baseline refers to the original project before the fork, i.e., Elasticsearch.

When investigating code quality further, Figure 5.7 unveils changes in code quality for each hotspot file from before the fork until one year after. For the common hotspot file (File 1), Elasticsearch shows negligible change, while OpenSearch exhibits modest improvement. Among files 2 through 10, hotspot files only in Elasticsearch, there is a general decline in code quality for both projects, with varying degree of severity. The only exception is file 5 which displays a slight increase in code quality for OpenSearch. On the other hand, file 5, 6, 7, and 8 experience substantial reductions in code quality for Elasticsearch. For files 11 through 17, hotspot files for OpenSearch, the outcomes are mixed, with some files experiencing declining

code quality while others remain unchanged or slightly improved. Overall, the analysis of hotspot files indicates divergent trends in code quality, with Elasticsearch files predominantly experiencing declines and OpenSearch files showing more variability.

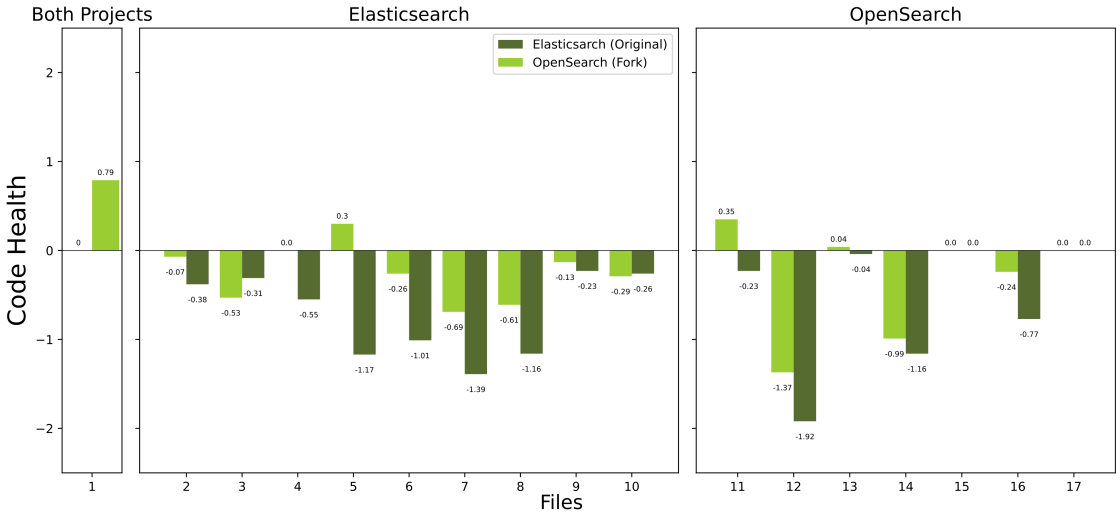


Figure 5.7: Changes in each Hotspot file compared to baseline one year after the fork. File 1 is a Hotspot file in both projects, while files 2-10 belonged to Elasticsearch and files 11-17 to OpenSearch.

5.1.4 Code Defects

This section presents the result related to code defects. It begins with the overall project followed by findings related to each project's hotspot file.

Summary of Section: Code Defects

Overall, both Elasticsearch and OpenSearch have seen an increase in bug frequency. However, following the fork, OpenSearch witnessed a notable decrease, with only one-third compared to its last data point in the baseline. Across all phases, OpenSearch consistently exhibits fewer bugs than both baseline and Elasticsearch. In the hotspot file analysis, Elasticsearch generally has fewer defects than OpenSearch.

InFigure 5.8, the bug frequency trends for Elasticsearch and OpenSearch are illustrated. The baseline reveals a steady rise in bug occurrences over time prior the fork. This upward trend persist for Elasticsearch (Original) post fork. However, OpenSearch (Fork) presents a notable deviation: immediately after the fork, the number of bugs drastically decrease from 1965 to 540. Despite this initial decrease, OpenSearch (Fork) maintains a comparatively lower bug frequency compared to Elasticsearch (Original), albeit gradually increasing over time.

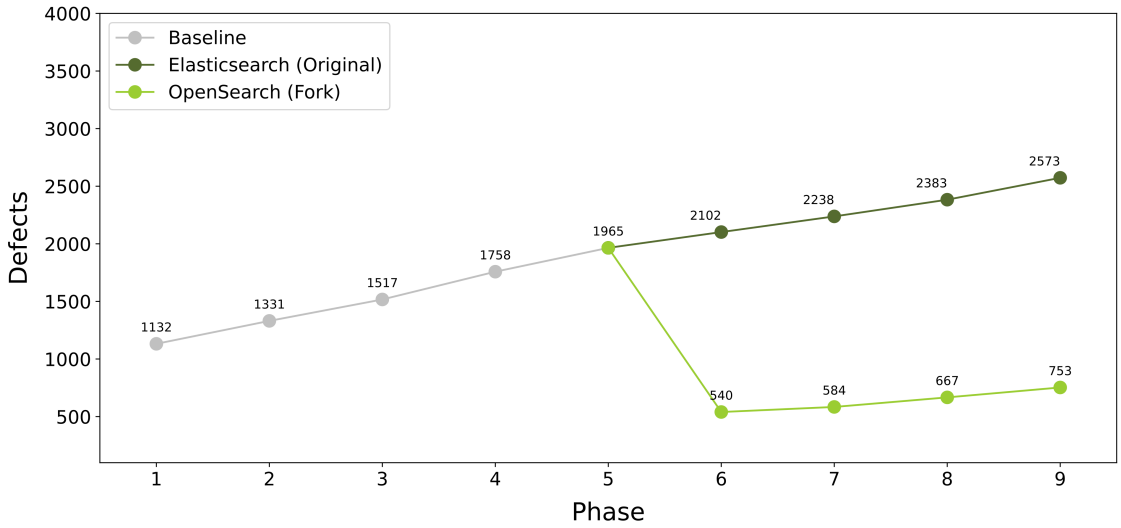


Figure 5.8: Figures over the defects for Elasticsearch and OpenSearch. Baseline refers to the original project before the fork, i.e., Elasticsearch

Figure 5.9 illustrates the defects found in hotspot files for Elasticsearch and OpenSearch. The figure reveals a significant difference in the number of defects between the two projects. Specifically, for files identified as hotspot files in both projects, Elasticsearch exhibited notably fewer bugs compared to OpenSearch. Among hotspot files within Elasticsearch, the frequency of defects was lower in all instances except one, where it matched OpenSearch's defect count. Conversely, among hotspot files found in OpenSearch, the frequency of bugs was higher in four out of seven files for OpenSearch compared to Elasticsearch.

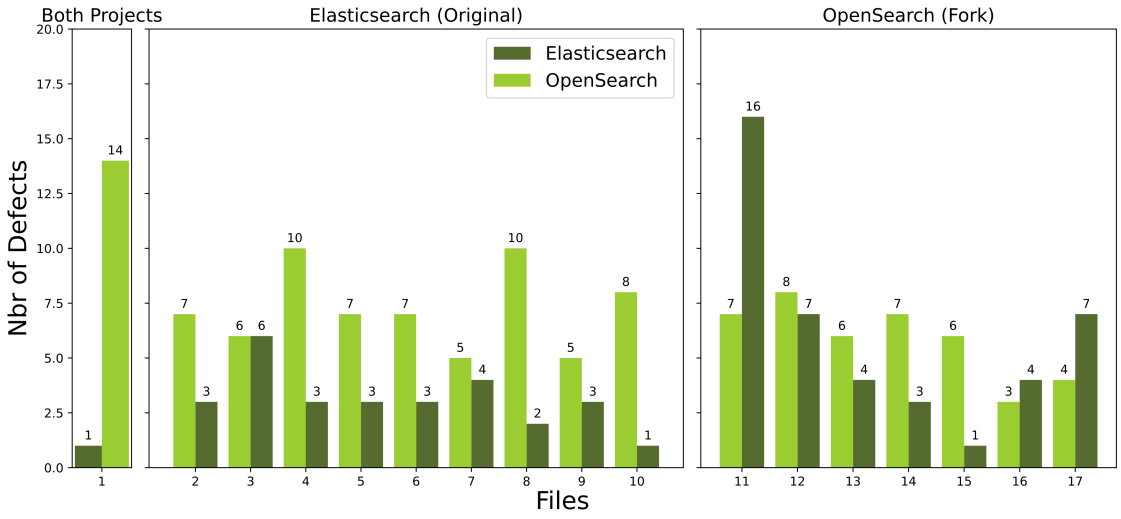


Figure 5.9: Changes in each Hotspot file compared to baseline one year after the fork. File 1 is a Hotspot file in both projects, while files 2-10 belonged to Elasticsearch and files 11-17 to OpenSearch.

5.2 Case B: ownCloud and Nextcloud

This section presents the results for Case B: ownCloud and Nextcloud. Each segment starts with a summary of key findings, followed by a detailed examination of the subject.

5.2.1 Contributors

This subsection addresses results on Key Contributors, Contributor Turnover Rate and Author Churn Trend.

Summary of Section: Contributors

In terms of key contributors, both projects exhibited similar characteristics. Prior to the fork, ownCloud witnessed an average increase of 3 contributors every 6 months. Following the fork, both projects encountered a reduction in contributors, with Nextcloud experiencing a more pronounced decline. Many contributors, e.g., Switchers, moved between the projects, with 49 switching from ownCloud to Nextcloud and 31 from Nextcloud to ownCloud. The Author Churn Trends showed some fluctuations in contributor counts for ownCloud, while Nextcloud maintained relatively stable post-fork.

Key Contributors

The findings concerning the selection of key contributors for Case B, as discussed in subsection 3.3.3 are presented in Table 5.2. The analysis revealed that ownCloud had 21 key contributors, constituting approximately 3,29% of the total, while Nextcloud had 18 key contributors, amounting to 6.77%.

Table 5.2: Data used to determine key contributors. 'Nbr of File Owners' and 'Nbr of File Owners ≥ 1%' represent the number of contributors of file owners in the project (both former and current) and the percentage compared to all contributors.

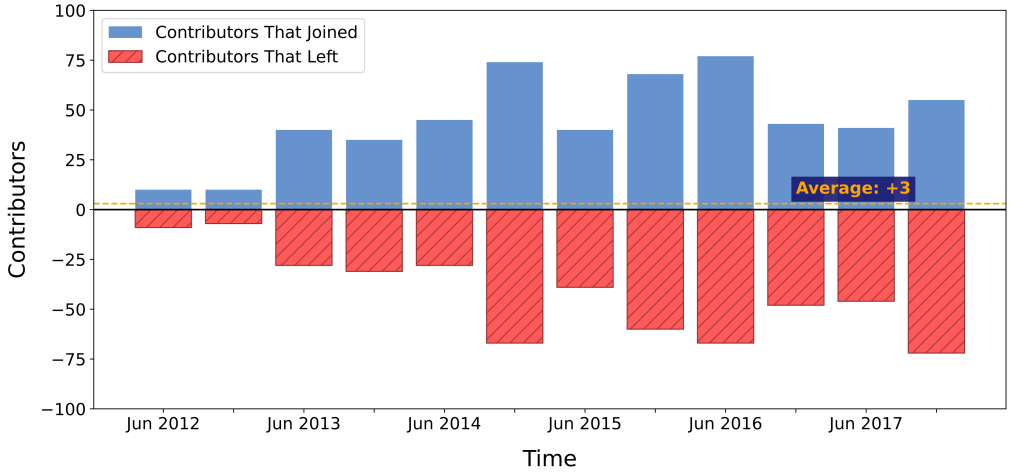
Project	Nbr of Files	Nbr of File Owners	Nbr of File Owners ≥ 1%
ownCloud (original)	4934	116 (18,14%)	21 (3,29%)
Nextcloud (fork)	3916	50 (18,80%)	18 (6,77%)

Contributor Turnover

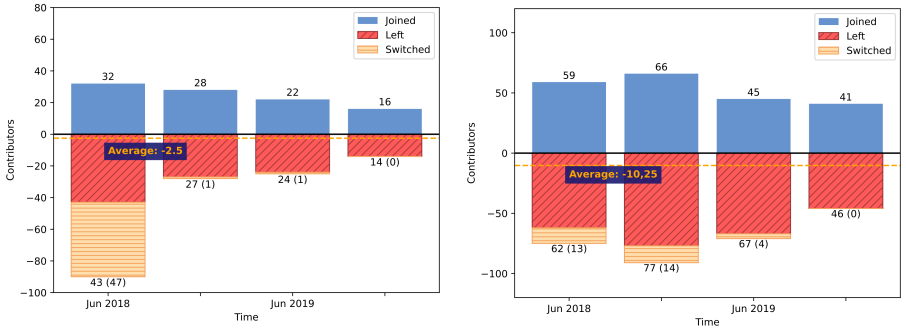
Regarding turnover rates, pre- and post-fork comparisons for ownCloud and Nextcloud are illustrated in Figure 5.10a. It reveals that before the fork, ownCloud experienced an average increase of three contributors per 6 months from June 2010 to June 2014. However, after June 2014, there was a shift towards more departures than new contributors joining the project. Following the fork, ownCloud displayed an average decrease of -2.5 developers per 6 months (see Figure 5.10b), while Nextcloud had an average decrease of -10.25 developers per 6 months

(see Figure 5.10c). It is worth noting that ownCloud exhibited a small increase in contributors in the last phase.

In terms of turnover rate of key contributors, prior the fork, 6 key contributors left ownCloud. During the first year post-fork, 7 key contributors left ownCloud, while Nextcloud suffered the loss of 2. Furthermore, during the second year post-fork, one key contributor left ownCloud, while 4 key contributors left Nextcloud.



(a) Baseline



(b) ownCloud

(c) Nextcloud

Figure 5.10: Turnover rate for (a) ownCloud before the fork, i.e., "Baseline", (b) ownCloud (original project) and (b) NextCloud (fork) after the fork.

Moving on to contributor flow between the projects, the analysis identified 85 contributors participating in the development of both ownCloud and Nextcloud. Among them, 12 key contributors were associated with ownCloud and 17 key contributors from Nextcloud. Regarding switchers, 49 contributors in ownCloud switched to working on Nextcloud, and 31 contributors in Nextcloud switched to working on ownCloud. 18 contributors were active in both projects, switching back and forth, and could not be considered active in only one project.

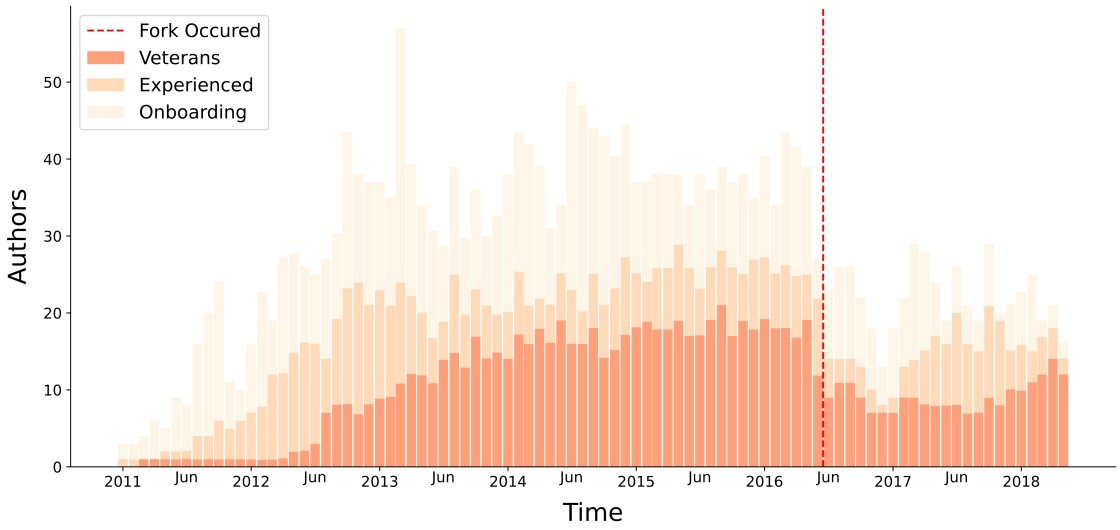
Concerning key contributors in ownCloud, 8 out of 12 were classified as Switchers, all transitioning to exclusive contributions to Nextcloud within six months of the fork. Additionally, three key contributors were considered dual developers during the first 18 months

post-fork, contributing primarily to ownCloud during that period. One key contributor was only regarded as a dual developer during the initial six months following the fork.

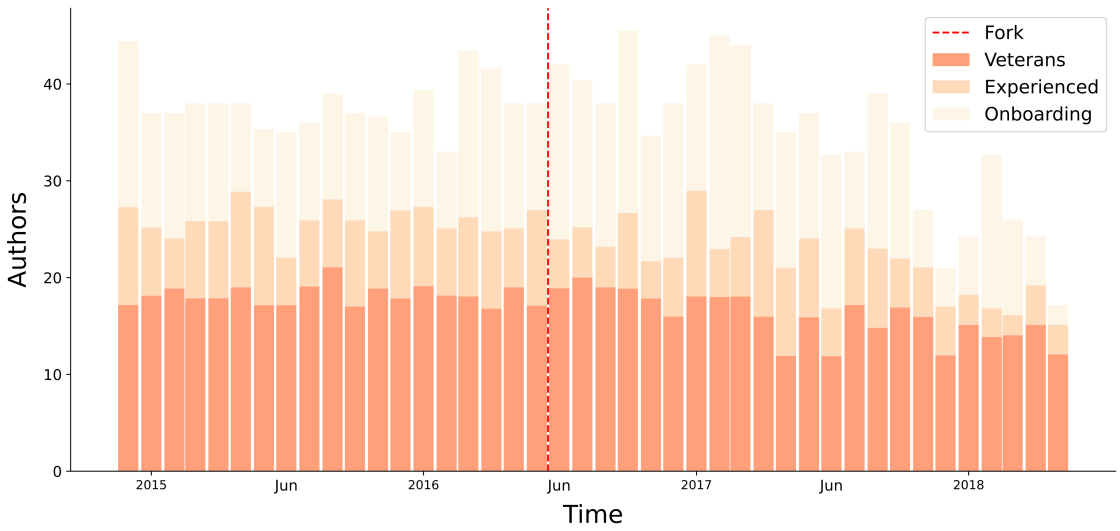
In Nextcloud, all key contributors had previously been active in ownCloud before the fork. However, among the 17 key contributors from Nextcloud, only 14 can be classified as Switchers. The remaining three engaged in dual development during the initial 18 months of the project.

Author Churn Trend

In Figure 5.12, the Author Churn Trends for both ownCloud and Nextcloud are presented, with individual sub-figures for each project.



(a) ownCloud



(b) Nextcloud

Figure 5.12: Author Churn Rate for each project in Case B: (a) ownCloud and (b) Nextcloud. Each bar represents six months, with the red dotted line indicating the date of the fork.

Beginning with Figure 5.11a, the trend for ownCloud unfolds. Starting with veterans, there was a steady increase of veterans prior to the fork. Post-fork, however, there is a dip in veterans, stabilizing and starting to increase again at the end of 2017. Similarly, the count of experienced contributors sees a steady climb followed by a stabilization prior to the fork. However, post-fork, there is a dip directly after, that stabilizes and increases again after six months. Lastly, in terms of onboardings, there was a gradual increase, punctuated by occasional peaks, prior to the fork. Post-fork, there is a decrease in onboardings compared to prior to the fork.

Turning to Nextcloud's Author Churn trend, depicted after the red dotted line in Figure 5.11b, alongside the baseline churn trend before the fork. Post-fork, Nextcloud maintains a relatively steady count of veteran contributors, mirroring the baseline project pre-fork. The number of experienced contributors shows minor fluctuations but remains relatively steady, with a slight downturn in 2018. Likewise, the count of onboardings remains stable, declining towards the end of 2017, with a slight uptick in early 2018 followed by another decrease.

5.2.2 Knowledge Loss and Code Familiarity

This section presents the findings regarding knowledge loss in the projects, followed by knowledge loss in hotspot files as well as the results regarding code familiarity.

Summary of Section: Knowledge Loss and Code Familiarity

Both projects experienced knowledge loss compared to baseline, with ownCloud (Original) having a higher knowledge loss than Nextcloud (Fork). Even two years after the fork, neither project manage to reach same levels as those observed in the baseline. Regarding the knowledge loss in the hotspot files, the results show variations in knowledge loss between the different projects, but what stands out is that Nextcloud has a low average knowledge loss in its hotspot files. Post-fork, the Code Familiarity decreases for both projects but is particularly pronounced for ownCloud.

In Figure 5.13 the knowledge loss for ownCloud (Original) and Nextcloud (Fork) is shown. The baseline for comparison references the original project before the fork. In Figure 5.13, a significant rise in average knowledge loss is evident for both projects post-fork. Notably, ownCloud (Original) experienced a particularly pronounced increase. Even after two years, the average knowledge loss for both projects had not stabilized to the baseline level.

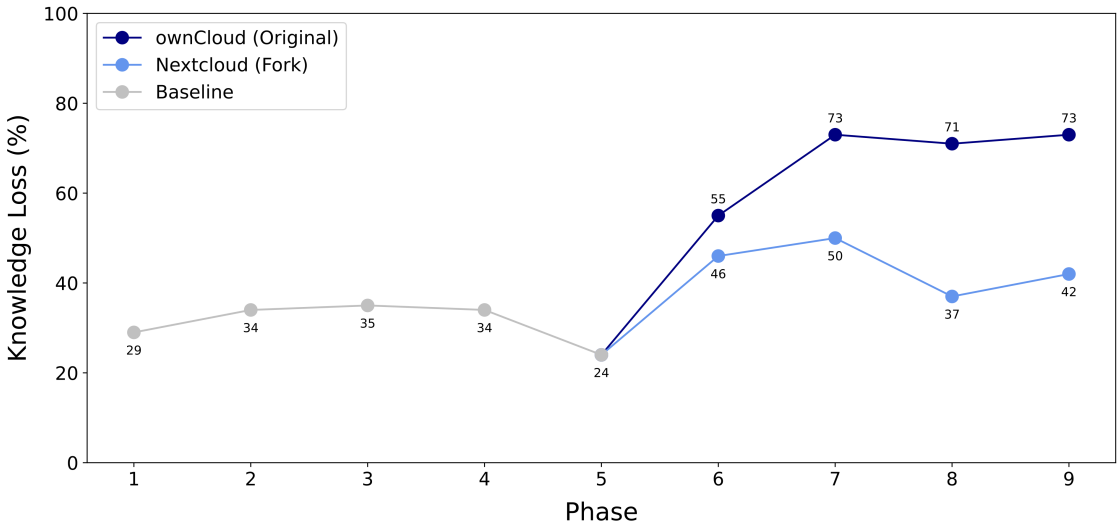


Figure 5.13: Average Knowledge Loss in ownCloud and Nextcloud. Baseline refers to the original project before the fork, i.e., ownCloud

Figure 5.14 provides a comparison of average knowledge loss observed in nine hotspot files within the ownCloud and Nextcloud projects, one year following the fork. Among these files, File 1 and File 2 are identified as hotspot files in both projects, while Files 3 to 7 are specific to ownCloud, and Files 8 and 9 are hotspot files in Nextcloud. The figure revealed variations in knowledge loss between the original and forked project. Overall, there is a diversity in the average knowledge loss across all hotspot files identified in both projects and within ownCloud specifically. Notably, Nextcloud has fewer hotspot files overall and exhibits lower average knowledge loss within these files compared to ownCloud. In contrast, among the hotspot files found in ownCloud, for two out of the five files ownCloud had a lower average knowledge loss compared to Nextcloud.

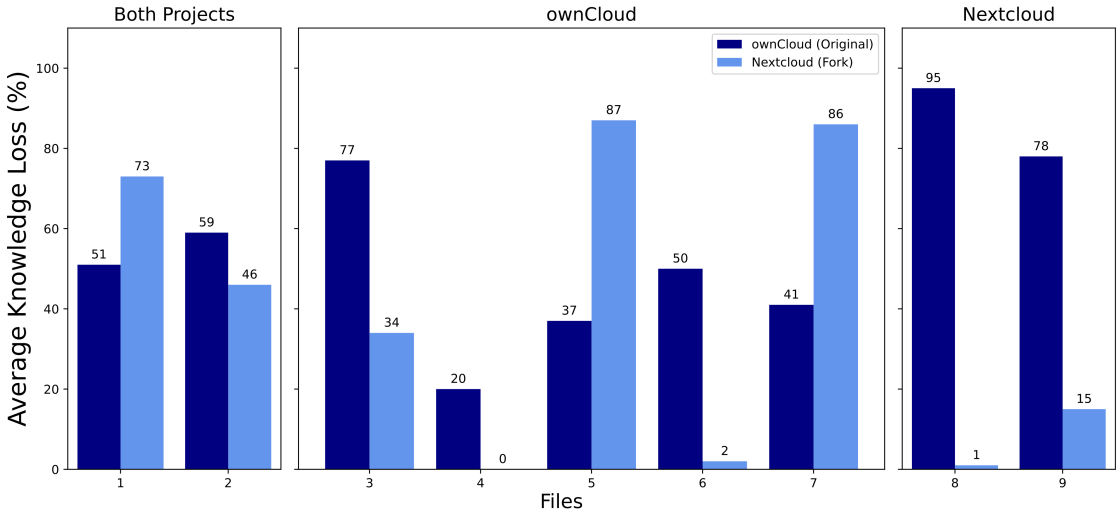


Figure 5.14: Average knowledge loss in hotspot files after the fork for Case B: ownCloud and Nextcloud.

In terms of code familiarity, Figure 5.15 displays the average code familiarity over time for ownCloud and Nextcloud. In Figure 5.15, it is evident that following the fork, there is a decline in code familiarity for both ownCloud and Nextcloud compared to the baseline. The decrease is particularly pronounced in ownCloud. Despite fluctuations, the trend suggests a persistent decrease in code familiarity over time for both projects.

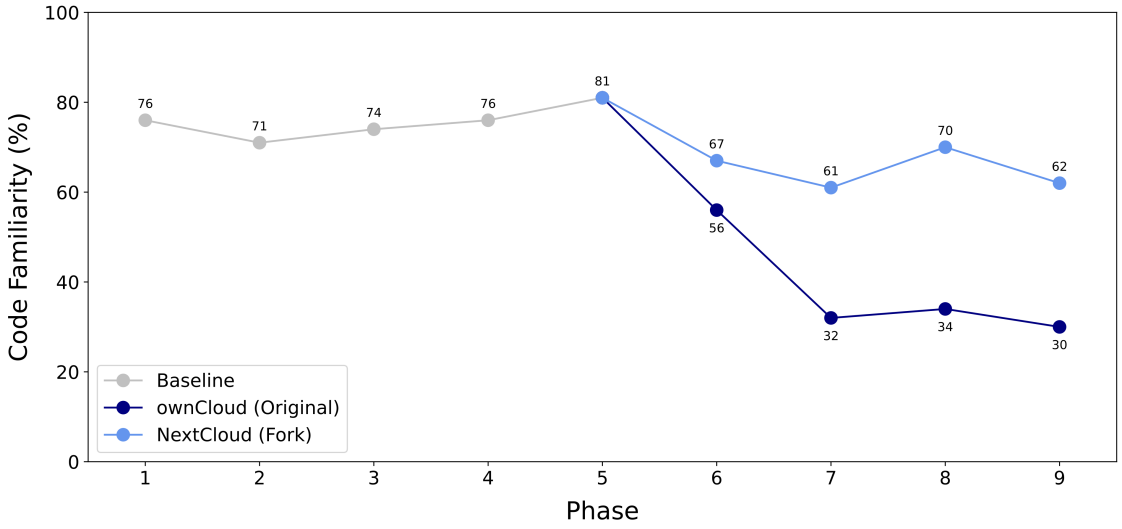


Figure 5.15: Average Code Familiarity for ownCloud and Nextcloud. Baseline refers to the original project before the fork, i.e., ownCloud

5.2.3 Code Quality

In this section, the results regarding code quality are presented. It includes code health results for whole projects followed by results regarding code health in hotspot files.

Summary of Section: Code Quality

The average code quality increased for both projects post-fork compared to the last phase for the baseline, as well as increased project sizes. Nextcloud demonstrates higher code quality than ownCloud across all phases. Regarding the code quality in hotspot files, the changes vary, but what is consistent is the increase in code quality for Nextcloud in the hotspot files for Nextcloud.

Figure 5.16 provides an overview of the code quality of both project, including the baseline, across all phases. Additionally, it presents the project size, measured in LoC. In Figure 5.16, an improvement in code quality for both projects post-fork is observed compared to the last measured value in the baseline. Also, there is a notable increase in project size for both projects post-fork. Notably, Nextcloud (Fork) consistently exhibits higher code quality across all phases. Both projects show similar sizes post-fork, except in the last measured phase, where ownCloud increased in terms of LoC.

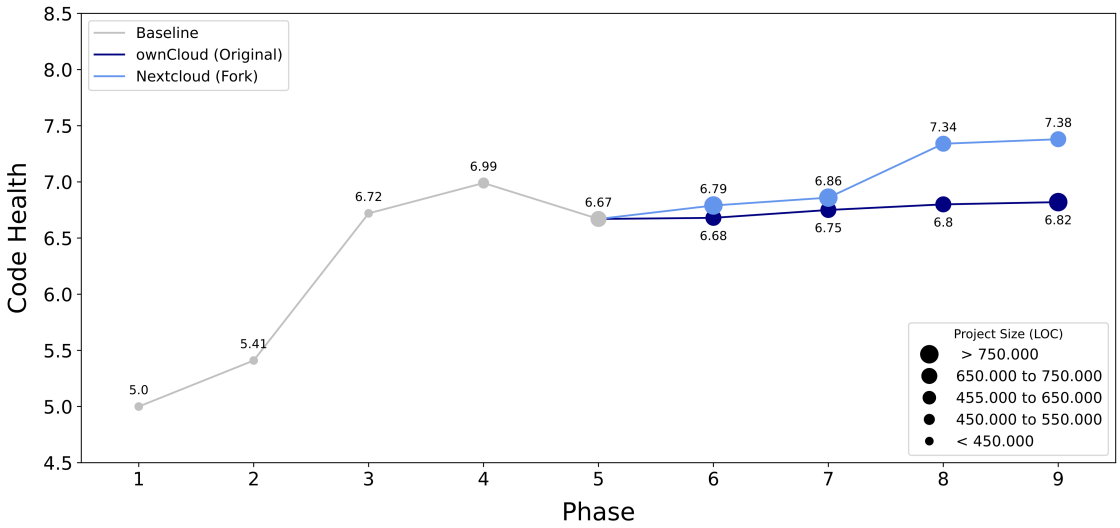


Figure 5.16: Code health and project size for ownCloud and Nextcloud. Project size is measured in lines of code (LoC). Baseline refers to the original project before the fork, i.e., ownCloud.

Figure 5.17 illustrates the shift in code quality from the baseline phase to one year post-fork for nine hotspot files across the ownCloud and Nextcloud projects. Files 1 and 2, present as hotspots in both projects. File 1 experiences a moderate improvement in ownCloud but a marginal increase in NextCloud, whereas File 2 undergoes significant deterioration in both projects. Files 3 through 7, identified solely as hotspots in OwnCloud, demonstrate diverse changes in code quality for both projects. Conversely, Files 8 and 9, hotspots files only in NextCloud, exhibit a distinct trend of enhancement in code quality for both files in Nextcloud, and a minor decrease for both files in ownCloud.

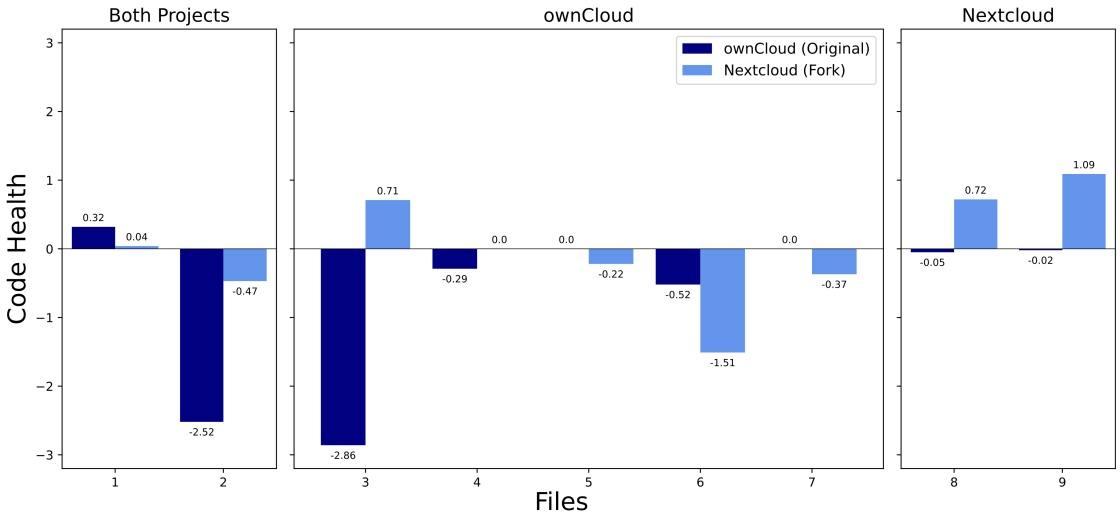


Figure 5.17: Changes in each hotspot file compared to baseline one year after the fork. Files 1 and 2 were hotspot files in both projects, while files 3-7 were hotspots in ownCloud and files 8 and 9 in Nextcloud.

5.2.4 Code Defects

This section describes the results regarding code defects, starting with bug frequency for the projects followed by the number of bugs found in the hotspot files.

Summary of Section: Code Defects

After the fork, ownCloud (Original) consistently exhibited a steady frequency of bugs across all phases. Conversely, Nextcloud (Fork) experienced an overall increase of bug frequency, except in the last phase, where it saw a decrease. Similar trends occur in the hotspot files: Nextcloud generally demonstrates a higher number of defects compared to ownCloud, except for file 6 where ownCloud has a higher rate, and file 8 where they both have the same rate. However, the analysis is limited by undefined defect counts in many files.

Figure 5.18 displays the frequency of bugs for both projects and the baseline. The figure reveals the prior to the fork, the frequency of bugs increases linearly up until the fork. After the fork, ownCloud (Original) has an almost steady frequency of bugs throughout the analysis period,, with a slight increase of 10 defects in each phase in the last two phases. Conversely, Nextcloud (Fork) has an increase of bugs after the fork, with an exception of the last phase, where it is a decrease.

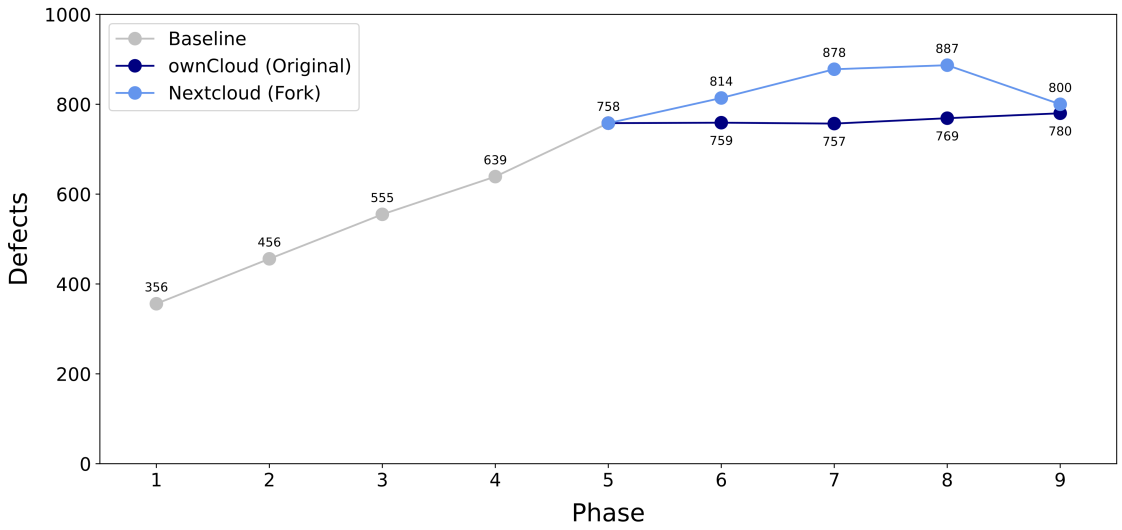


Figure 5.18: Figures over the defects for ownCloud and Nextcloud. Baseline refers to the original project before the fork, i.e., ownCloud

Regarding defects within hotspot files for ownCloud and Nextcloud, it is important to acknowledge a significant limitation in this figure: a notable portion of files for both projects lack defined defect counts, potentially skewing the analysis results. Given this inconsistency, for a fair comparison, attention will be focused on file 5 and file 8. Nextcloud consistently exhibited a higher defect incidence, except for two cases: file 6 and file 8. As illustrated in Figure 5.19, there is no data for file 6 in Nextcloud. In file 8, it shared an equal number of defects with ownCloud.

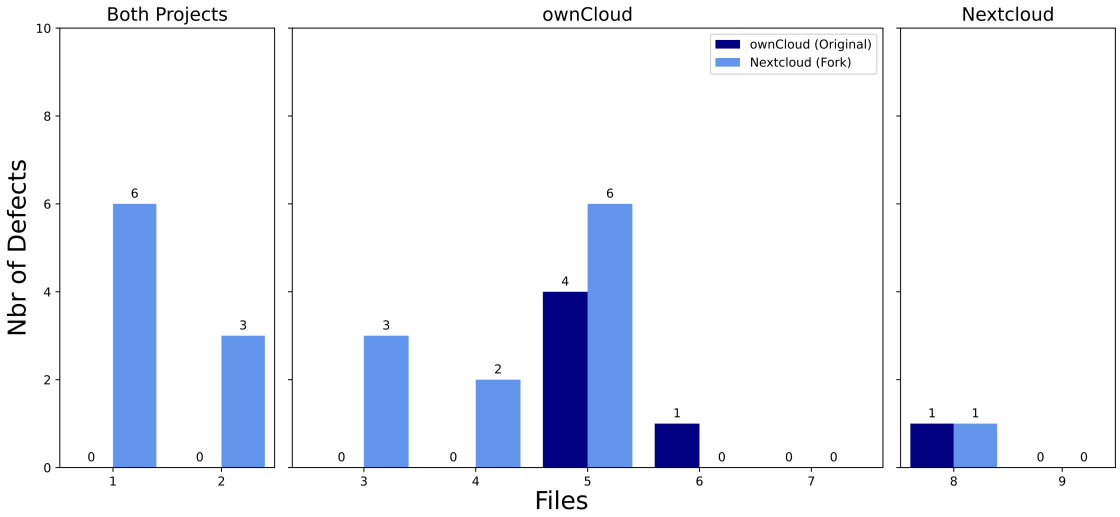


Figure 5.19: Changes in each hotspot file compared to baseline one year after the fork. Files 1 and 2 were hotspot files in both projects, while files 3-7 were hotspots in ownCloud and files 8 and 9 in Nextcloud.

5.2.5 Interviews

This section presents the outcomes of interviews carried out with contributors involved in Case B projects. It commences with a concise summary of the interview results, followed by a brief overview of the interview process. Subsequently, it delves into a presentation of the responses, organized around three themes: Motivation behind the fork, Relationship between the projects and impact on the code base. To ensure privacy, author names have been anonymized and replaced with labels such as Author A, Author B, etc.

Summary of Section: Interviews

The two interviewed authors, who contributed to ownCloud before the fork and Nextcloud after the fork, both mentioned strategic issues as part of the reason behind the fork, as well as political reasons and tension between the founders of ownCloud. It was mentioned that most of the core team of developers made the move from ownCloud to Nextcloud, and there was no communication or helpful relationship between the projects. Despite the minimal knowledge loss in Nextcloud, the project management was different and described as more chaotic, resulting in more bugs. One author mentioned that as a result of ownCloud having more knowledge loss they were developing their code base less to mitigate this problem.

In total, two interviews were conducted. The interviewees will be referred to as Author A and Author B. Both interviewees had been active in ownCloud prior to the fork and Nextcloud after the fork. However, only one of the interviews, Author A, is still active in Nextcloud today.

Motivation behind the fork

Author A highlights political and strategic issues as motives for the fork, citing a discomfort within ownCloud's engineering team regarding the company's direction. Author B expands on this motive, describing additional tension between the founders of ownCloud, along with the awareness that ownCloud might be acquired by another company at that time. Author B also confirms the discomfort within ownCloud's engineering team, stating that after the announcement, *"Within a week, almost all core developers sent in their letter of resignation [...]"*.

Relationship between the projects

Author A stated that *"There was quite some one-direction tension"*, as well as *"No direct communication between the projects"*. Author B pointed out that post-fork, there was no sharing or utilization of code between the projects, largely due to licensing issues. As Nextcloud operates under the AGPL, any entity seeking to integrate features or developments by Nextcloud would be required to enclose their entire codebase. As told by Author B, ownCloud did not want to enclose their enterprise code, hence refrained from adopting any code developed by Nextcloud to avoid having to reveal proprietary code.

Impact on the code base

Both authors assert that the Nextcloud project suffered minimal knowledge loss as a result of acquiring the majority of the contributors in ownCloud. Additionally, as a way to handle the knowledge loss in ownCloud, Author B speculated that ownCloud suffered minimal effect from the knowledge loss due to less development on their code base after the fork, noting that *"New releases of ownCloud looks almost the same as it did before the fork"*.

In terms of project management, the changes were more conspicuous. Author B remarked that ownCloud operated in a less "chaotic" manner compared to Nextcloud. Furthermore, it was noted that many of the individuals who joined Nextcloud were not accustomed to structured workflows, which may have contributed to this perception. Additionally, the Author mentioned a lack of proper road mapping and a more "yolo" work style in Nextcloud. Author A mentioned, "In various areas, there was quite a backlog with taking opened community contributions. [...] There were 30 features merged soon after as the community [...] developers happily switched to the Nextcloud [...] project and took the lead on it."

Additionally, Author B described that the processes of bug reports and QA assurance differed between the projects. "In ownCloud, the QA was paid by the company, but in Nextcloud there had to be new processes," and "They had other processes than ownCloud with, for example, bug reports; the engineers had to do all of that in Nextcloud."

In terms of defect management, Author B observed a significant increase in bugs and noted that many new releases in Nextcloud actually were not ready to be released.

Chapter 6

Discussion

This chapter discusses the results and methodology for this thesis. It begins with a figure showing the summary of the results, followed by a discussion of the results, outlined in chapter 5, and a discussion about the methodology, outlined in chapter 3.

The summary of the results gathered in this thesis is presented in Figure 6.1. This figure provides an overall combination of the results from the two years following the fork. Each research question's respective section will include a discussion of the figure.

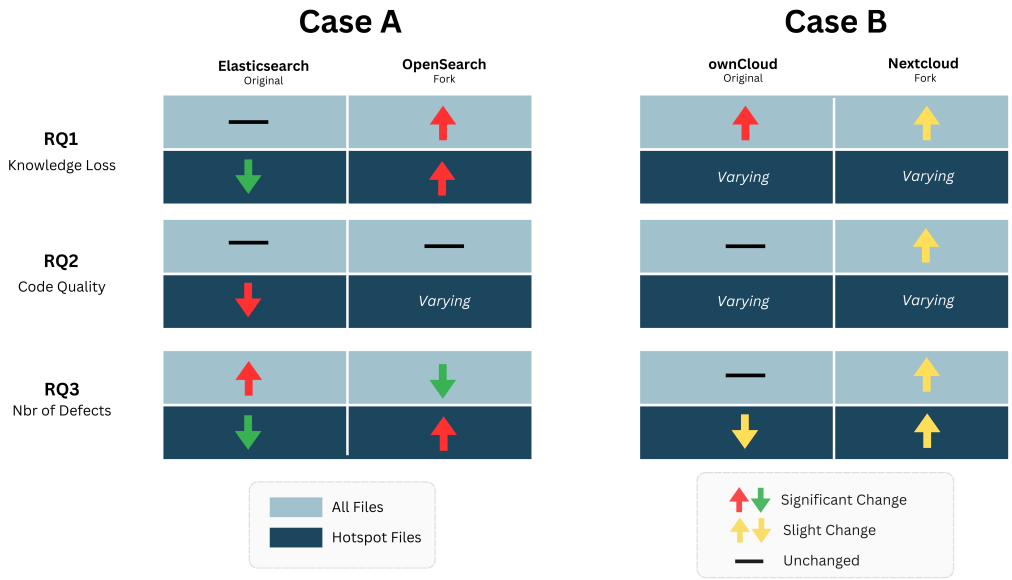


Figure 6.1: Summary of the results. Varying results refer to results that displayed no clear trend or pattern.

6.1 How can the Forking Knowledge Loss be Characterized? (RQ1)

This section discusses the findings concerning research question 1. It begins with a summary of the main points discussed, followed by a discussion about Knowledge Loss Duration and Knowledge Loss Scope.

RQ1: How can the knowledge loss resulting from the forking be characterized in terms of 1) Knowledge Loss Duration and 2) Knowledge Loss Scope?

In terms of knowledge loss scope, as shown in Figure 6.1, Case A showed minor knowledge loss (29%-40%) in the original project (Elasticsearch) and significant loss (79%-90%) in the forked project (OpenSearch), attributed to the absence of switchers and difference in code base sizes. In case B, the original project (ownCloud) experienced higher knowledge loss (55%-73%) compared to the forked project (Nextcloud) with a lower loss(42-50%), likely due to significant migration of key contributors and more active development in the fork.
Regarding knowledge loss duration, as seen in Figure 6.1, in case A, the original project (Elasticsearch) suffered almost none, while the forked project (OpenSearch) continued to experience high loss for at least two years. In case B, both projects sustained high knowledge loss for at least two years.

Case A

The findings revealed an intriguing disparity between the outcomes of the two cases under study. In Case A, the forked project (OpenSearch) showed 50% more knowledge loss than Elasticsearch (original), as seen in Figure 5.3. This pattern is similarly reflected in the results for code familiarity; however, the difference is more pronounced. The original project had roughly 57% higher code familiarity compared to the forked project (OpenSearch). Again, across all hotspot files, the original project (Elasticsearch) had a lower knowledge loss compared to the fork (OpenSearch). Thus, in terms of knowledge loss scope, the forked project (OpenSearch) had a notably higher knowledge loss, as seen in Figure 6.1.

When comparing these findings with the contributor turnover, a distinct trend emerges. As shown in Figure 5.2, the original project (Elasticsearch) remains relatively unaffected by the fork, while the forked project (OpenSearch) experiences a significant decrease and fails to stabilize to pre-fork levels within the analysis period. This trend is further evident in the number of active authors per months, with the forked project (OpenSearch) having roughly one-third of active authors compared to the original project (Elasticsearch). Contradictorily, the forked project (OpenSearch) has a higher contributor turnover rate post-fork, mostly involving new developers, aligning with the findings in the author churn trend.

Thus, the absence of contributor turnover, or rather, the absence of switchers from the original projects, may explain the higher knowledge loss and lower code familiarity seen in the forked project. Given a sizable codebase of approximately 1,4000,000-1,800,000 lines of code, taking ownership of all files could take time. While the forked project (OpenSearch) does exhibit a decrease in knowledge loss and an increase in code familiarity throughout

the analysis period, the presence of only two key authors and a considerably low number of active authors per month implies that re-establishing ownership over the codebase could be a protracted process.

Regarding knowledge loss duration, Case A suggest that it may persist beyond the analysis time frame due to limited active contributors and slow re-establishment of codebase ownership. This prolonged duration, particularly in the forked project (OpenSearch), might hinder its progress compared to the original project (Elasticsearch).

Case B

In case B, the original project (ownCloud) instead showcased a higher average knowledge Loss. In this case, the forked project (Nextcloud) had an average of roughly 22% lower knowledge loss compared to the original project (ownCloud) post-fork, see Figure 5.13. However, both projects experienced an expansion in lines of code compared to the baseline (ownCloud prior to the fork). Similar trends emerge when considering code familiarity. The forked project (Nextcloud) demonstrated an average code familiarity approximately 27% higher than the original project (ownCloud) post-fork. Yet, like knowledge loss, both projects experienced a decline in code familiarity compared to the baseline. Examining the average knowledge loss in hotspot files yields more nuanced results. Notably, across all hotspot files found in the forked project (Nextcloud), the average knowledge loss was significantly lower compared to the original project (Owncloud).

In Case B, the contributor turnover results contrast with those of Case A. The maturity of the forked project, Nextcloud, remains unaffected by the fork, see Figure 5.12, unlike the decline seen in the original project, ownCloud, although this decline is less severe than in Case A. Nextcloud has more active authors per month than ownCloud, though both see a post-fork decrease. The contributor turnover graph shows both projects with a decreasing average number of authors, with Nextcloud's decline (-10.25) being much steeper than ownCloud's (-2.5) every six months.

A notable factor is the high number of switchers: 47 authors moved from ownCloud to Nextcloud within six months post-fork, including eight key contributors. Overall, 12 key contributors, or 57% of ownCloud's key contributors, switched to Nextcloud, aiding its smooth transition. In the first year post-fork, ownCloud lost seven key developers, contributing to increased knowledge loss from phases 5 to 7. In the second year, four key developers left Nextcloud, leading to a slight increase in knowledge loss in the final phase.

These findings regarding Case B align with the findings of Gamalielsson and Lundell [33] when they studied the long-term sustainability of forked OSS projects. They found that the forked project they looked at (LibreOffice) managed to be long-term sustainable by attracting committers from the original project (OpenOffice.org). Most of the contributors in the fork came from the original project ensuring continuity of knowledge. This is similar to the findings regarding ownCloud and Nextcloud, where the fork (Nextcloud) managed to be sustainable by attracting developers from the original project (Nextcloud).

These findings are consistent with those obtained from the interviews. Both interviewees noted that a majority of the key contributors in the original project (ownCloud) transitioned to the forked project (Nextcloud). Additionally, they expressed that there was no noticeable loss of knowledge experienced by the forked project post-fork. However, as the graph over average knowledge loss displayed, the forked project did in fact experience an increase in

knowledge loss, but not to the same extent as the original project (ownCloud).

Conclusion

The study by Nassif and P. Robilard on Turnover-Induced Knowledge Loss, which builds upon the research by P.C. Rigby et al. [70], discovered that files written by leavers often remained abandoned for extended periods [53]. This observation aligns with the findings that the duration of knowledge loss tends to be lengthy, often lasting at least two years, as observed in both Case A and Case B.

As summarized in Figure 6.1, Case A exhibits a prolonged duration and greater scope of knowledge loss in the forked project. In contrast, Case B demonstrates a reduction in the scope of knowledge loss in the forked project compared to the original. However, the duration of knowledge loss in Case B remains comparable to that observed in Case A.

6.2 What is the Association Between Forking and Code Quality? (RQ2)

This section discusses the findings related to research question 2. It begins with a summary of the main points discussed, followed by a discussion and comparison of the results of code quality in both cases.

RQ2: How is the knowledge loss from forking associated with changes in code quality?

There is no strong indication of an association between knowledge loss after forking and changes in code quality, therefore alternative reasons for changes in code quality were discussed. The other factors discussed were project size, number of active contributors and contributor turnover. The findings were that there might be an association between project size and code quality, but it is not the only factor, the number of active developers did not seem to have an impact and there might be a relationship between the turnover of experienced contributors and the code quality, however only seen in one case. There is no single factor that consistently is associated with changes in code quality, and it appears that a combination of factors, including project size and contributor turnover collectively influence the change in code quality.

Impact of Forking on Code Quality

In terms of code quality, both cases showed varying results, as shown in Figure 6.1. In Case A, both projects displayed nearly identical code quality throughout all analysis phases following the fork, with a marginal decrease of 0.26 in average code health. In terms of hotspot files, the forked project (OpenSearch) showcased better code health in nearly all files compared to the original project (Elasticsearch). In Case B, both projects experienced an increase in code quality, yet the forked project (Nextcloud) demonstrated a more substantial improvement. This pattern also occurred in the hotspot files. The forked project (NextCloud) exhibited better

code quality in its own hotspot files as well as in those shared with the original project. Overall, the forked project demonstrated better code quality compared to the original project, except for three files identified as hotspot files in the original project (ownCloud).

Effect of Code Base Size on Code Quality

Given the absence of a noticeable correlation upon comparing these two cases, it prompts a reflection on potential underlying causes and implications. One possible explanation is the fluctuation in project sizes. Larger projects may struggle with code quality due to complexity, while downsizing might improve it by removing poorly written code. In Case A, Elasticsearch grew in size while OpenSearch shrank, yet both maintained similar code quality throughout. In Case B, despite the size increase in Nextcloud, they consistently exhibited higher code quality than ownCloud, and further improved in quality when Nextclouds size later decreased. These observations suggest a potential link between project size and code quality, though size alone is not the sole influencing factor; other elements likely play significant roles.

Influence of Active Contributors on Code Quality

The number of active contributors might influence code quality, as larger communities can enhance code review, bring diverse perspectives, and expedite bug fixes and feature development. However, they can also introduce challenges like communication overhead and integration complexity. In Case A, Elasticsearch initially had more active users post-fork, which decreased over time, while OpenSearch started with fewer active users but saw an increase. Despite these differences, both projects maintained similar code quality, suggesting active contributor count is not a significant factor here. In Case B, Nextcloud had more active contributors than ownCloud and exhibited higher code quality. Although the number of active contributors in both projects decreased over time, Nextcloud's code quality continued to improve. This indicates that while active contributor count may influence code quality, it is not the sole determining factor.

The Impact of Contributor Turnover and Maturity on Code Quality

When discussing the effect contributors' turnover can have on code quality, the project experience of the contributors should also be acknowledged. In Case A, Elasticsearch maintained project maturity and retained all key contributors post-fork, while OpenSearch saw a drop in maturity and had fewer key authors but still matched Elasticsearch's code quality. This suggests that contributor turnover alone doesn't fully determine code quality in Case A. In Case B, Nextcloud maintained project maturity and retained all key contributors, whereas ownCloud declined in maturity and lost many key contributors. Nextcloud demonstrated higher code quality than ownCloud, with both improving post-fork. This indicates that experienced contributor turnover significantly affects code quality, but other factors also play a role, as evidenced by Nextcloud's lack of increased maturity despite better code quality.

Conclusion

In conclusion, the continuity of knowledgeable contributors is crucial for maintaining code quality. In Case A, Elasticsearch retained its key contributors, allowing it to expand without compromising quality. Similarly, in Case B, Nextcloud retained all key contributors and showed higher code quality and maturity than ownCloud. Projects with lower maturity levels managed to maintain similar or slightly lower code quality. OpenSearch, despite reduced maturity and size, matched Elasticsearch's code quality. Nextcloud improved its code quality by decreasing in size, possibly by removing poor-quality code, indicating that downsizing can enhance code quality. However, ownCloud maintained its code quality without downsizing, possibly due to reduced ongoing development. This suggests that multiple factors contribute to maintaining code quality.

6.3 What is the Association Between Forking and Bug Reports? (RQ3)

This section discusses the findings related to research question 3. It begins with a summary of the main points discussed, followed by a discussion and comparison of both cases.

RQ3: What is the association between forking, particularly in scenarios involving significant knowledge loss, and the frequency of bug reports in software modules subsequent to the forks?

There does not seem to be any association between forking and the frequency of bugs, particularly in scenarios involving significant knowledge loss, and more knowledge loss does not imply that the frequency of bugs will increase, as shown in Figure 6.1. The different cases showed different results, where the project that suffered more knowledge loss in case A had fewer bugs, and in case B the project with more knowledge loss had more bugs. Project size was discussed but not found to be a direct cause, but rather a contributing factor and other factors seem to play a bigger role in the frequency of bugs. For example, how much development is made in the code, management of features and releasing code that is not ready to be released are some reasons that were found that might have had an impact on the frequency of bugs.

Impact of Forking on Code Defects

In Case A, the forked project (OpenSearch) displayed a notable decrease in bug reports. After the fork, bug reports for the forked project (OpenSearch) decreased to about one-third of those seen in the original project (Elasticsearch). However, in subsequent analysis periods, both projects saw an increase in bugs. The forked project (OpenSearch) experienced an average rise of approximately 20 bugs every six months, while the original project had a larger increase of around 150 bugs per six-month period. Conversely, in terms of hotspot files, the forked project (OpenSearch) consistently showed either equal or higher bug frequency across almost all hotspot files. Only in three files, the forked project (OpenSearch) exhibited a lower frequency of bugs, all of which were specific to the forked project.

In Case B, the forked project (Nextcloud) showed a higher frequency of bugs compared to the original project (ownCloud). This frequency increased during all three phases post-fork, except for the last phase where it decreased. In contrast, the frequency of bugs for the original project (ownCloud) remained largely consistent throughout all post-fork phases. Upon investigating the results for the hotspot files, a similar pattern emerged. The fork consistently exhibited a higher or equal amount of bug frequency compared to the original project. During the interviews, similar trends regarding bug occurrence were reported. One interviewee noted that the frequency of bugs was higher in the forked project and highlighted a lack of structure to prevent bugs. Another interviewee mentioned that numerous changes were merged into production in the forked project immediately after the fork, suggesting potential challenges in managing code quality during the transition period.

Project Size Impact on Code Defects

When considering potential causes and effects for these results, one plausible explanation could be attributed to changes in project size. One of the findings by Tornhill and Borg was that for files that have a code health of 1, the probability of that file being very big is high [4]. Typically, an increase in project size implies increased complexity and potentially more opportunities for bugs. However, the nature of these additions must also be considered: well-structured and thoroughly tested changes might not necessarily lead to a proportional increase in bug frequency, while hastily and poorly tested implementations could introduce new bugs into the code base. In Case A, the original project (Elasticsearch) exhibited a higher frequency of bugs, coinciding with an increased project size compared to the forked project (OpenSearch). Moreover, in Case B, the forked project (Nextcloud) displayed a higher frequency of bugs alongside a slight increase in project size. This observation supports the assumption that increased project size potentially amplifies the occurrence of bugs. Yet, in the final analysis phase in case B, both projects exhibited the same number of bugs despite having opposite project sizes - the original project (ownCloud) increased, while the forked project (Nextcloud) decreased. Thus, in these cases, the correlation between project size and bug frequency should rather be interpreted as a contributing factor rather than a direct cause. As mentioned above, it is also important to acknowledge the nature of the changes introduced. As stated in the interviews, the forked project (Nextcloud) lacked a clear approach to handling quality assurance and bugs. Therefore, one possible explanation for the decline in bugs during the last phase of Case B could be the result of implementing better bug management practices.

Correlation between Knowledge Loss and Code Defects

Another plausible explanation for these results could be the correlation between knowledge loss and code defects. When a project undergoes forking, there could be a loss of institutional knowledge as developers who were familiar with the original codebase may no longer be involved or may have limited involvement. This loss of knowledge might lead to a misunderstanding of the codebase, misinterpretations of design decisions, and an overall decrease in code quality, potentially resulting in an increase in code defects. Additionally, the introduction of new contributors who may not be familiar with the new codebase can further exacerbate the situation. These may inadvertently introduce bugs while making changes or

may struggle to identify and fix existing issues due to a lack of familiarity with the code.

In Case A, the original project (Elasticsearch) showcased a higher frequency of bugs, yet, a lower knowledge loss. Whereas the forked project (OpenSearch) showcased the opposite, see Figure 6.1. Conversely, in Case B, both projects experienced increased knowledge loss, with the original project (ownCloud) undergoing the most significant loss and also showing the highest frequency of bugs. Hence, findings in case B conform to the assumption made between the frequency of bugs and knowledge loss. However, the findings in Case A present somewhat of a paradox, contradicting the conventional expectations, as higher knowledge loss correlates with lower bug frequency. One potential explanation for this correlation could be that as contributors leave, there is a reduction in the number of changes made to the code base – thus, lowering the potential risk of introducing new bugs. However, while knowledge loss may contribute to bug frequency, as shown in these two cases, it does not apply to all cases, and therefore should not be considered a direct cause.

Association between Code Quality and Code Defects

Regarding the study by Tornhill and Borg that found that there are 15 times more bugs in low-quality code than in high-quality code [77], the same correlation was not found in this thesis. However, none of the projects analyzed in this thesis had a code health score in the unhealthy category. In Case A both projects had almost the same code health, but different numbers of bugs, and in Case B the project with the better code quality also had more bugs.

Conclusion

In conclusion, the findings from both Case A and Case B suggest that there is no clear association between knowledge loss and the frequency of bugs in scenarios involving forking. While certain correlations were observed, such as a higher frequency of bugs coinciding with higher knowledge loss in case B, as well as project size influencing the occurrence of bugs, these patterns are not applicable to both cases. Thus, while knowledge loss may contribute to bug frequency in certain cases, it cannot be considered a direct cause based on these two cases.

6.4 How can Knowledge Loss from Forking be Mitigated? (RQ4)

This section discusses the findings related to research question 4. It begins with a summary of the main points discussed, followed by a discussion and comparison of both cases.

RQ4: How did the OSS communities mitigate the knowledge loss from forking?

Given the lack of data, determining how communities have mitigated potential knowledge loss is significantly constrained. In Case A, only the forked project experienced significant knowledge loss. Despite having a positive trend in onboarding and retaining experienced developers, knowledge levels remain below those observed pre-fork. In Case B, while both projects experienced knowledge loss, Nextcloud (fork) developers claimed no impact and thus did not implement any mitigation measures. Owncloud's (original) approach remains unknown. According to the switchers, ownCloud addressed their knowledge loss by halting the development.

To gain insight into how the OSS communities mitigated the potential knowledge loss from forking, interviews would serve as the primary source of data. Therefore, qualitative data were obtained through interviews, helping to answer the fourth research question. However, getting contributors to participate proved more difficult than anticipated. Only two people agreed to share their insights, both of whom were switchers who contributed to the same project throughout the four-year analysis period. Therefore, the insights gained from the interviews were predominantly from one side of the fork's narrative and limited to main insights into one project.

Case A

In case A, the average knowledge loss in Elasticsearch does not change much compared to before the fork. This suggests that there was no significant knowledge loss, and therefore the community connected to Elasticsearch did not have to employ any strategy to mitigate knowledge loss. On the other hand, the forked project, OpenSearch, suffered knowledge loss after the fork. Going from 90% average knowledge loss after the fork, they gradually regained some knowledge over time, and after two years, the average knowledge loss was down to 79%. Unfortunately, no developer wanted to partake in an interview and therefore it is difficult to know exactly what they did to mitigate the knowledge loss. However, a noteworthy observation is that, despite appearing to receive significant support from industrial corporations prior to the fork, OpenSearch, the forked project, still experienced significant challenges in terms of project maturity, as seen in Figure 5.2b. However, a closer examination of the figure reveals a positive trend: the number of onboarded contributors and their experiences increased steadily over time for OpenSearch. This indicates a growing community of contributors who remained active for 6-12 months. This increase of more experienced or even veteran contributors could help the project regain and retain their knowledge of the code base.

Case B

In Case B, according to a developer in Nextcloud during an interview, one suggested way that ownCloud handled the knowledge loss from forking was that they stopped developing the code. This strategy does not mitigate the knowledge loss since it does not mean they regained knowledge about the code. But without being able to interview developers who contributed to ownCloud after the fork it cannot be ruled out that they did something to mitigate the

knowledge loss. However, as shown in Figure 5.13 ownCloud does not regain knowledge the two years following the fork. This suggests that ownCloud did not manage to mitigate the knowledge loss. For the forked project, Nextcloud, the average knowledge loss increased compared to the baseline. When listening to the key developers during the interviews they insisted that they did not suffer any knowledge loss that affected the development of the code. Therefore they said that they did not have to do anything to mitigate the knowledge loss.

Conclusion

In conclusion, in Case A, where OpenSearch was the project that mainly suffered knowledge loss managed to mitigate this and decrease it over time, although how the community did that is unknown. One way could be that they managed to retain experienced developers. In Case B, where both projects' average knowledge loss increased, according to developers in Nextcloud they did not suffer from this knowledge loss. Therefore, they did not do anything to mitigate this loss. For ownCloud, who suffered more knowledge loss, how they mitigated this loss is unknown According to developers who switched from ownCloud to Nextcloud, they handled the loss by stopping the development of their code base.

6.5 Limitations and Threats to Validity

This section discusses potential threats to the validity and limitations of this thesis as well as methods employed to mitigate them, as described in chapter 3. It addresses construct validity, internal validity, external validity and reliability [72].

6.5.1 Construct Validity

To mitigate potential threats to construct validity, various strategies have been employed, including data triangulation, operationalization of constructs, and standardization of data collection procedures. However, despite these precautionary measures, the threat to construct validity still remains.

One particular concern is data biases, i.e., incomplete or inaccurate data. For instance, configuring the analysis accurately is essential to retrieve correct data. In Codescene, the contributors' emails or names are used to identify their contributions and calculate knowledge metrics [12]. However, contributors may use different aliases, leading to data bias and inaccuracy [15]. CodeScene offers alias merging based on email addresses to reduce bias. However, connecting aliases to the same email address is not always possible, potentially resulting in one contributor being counted as multiple individuals. This affects accuracy in knowledge metrics and analysis outcomes, including identifying leavers and joiners. The issue extends beyond individual projects to comparisons between projects, especially in identifying switchers. Thus, if a developer uses different names, GitHub accounts or emails for different projects, they will not be recognized as switchers.

Another concern is the purposive sampling method employed to identify key contributors and potential interviewees also raises concerns regarding bias. Although it limited the

participant selection, it was necessary for interviewees to possess adequate project knowledge. The sampling group's size was not small due to few interviewees, but rather due to low interest. The possibility of less-than-friendly relationships between projects might deter developers from participating, and alternative methods like surveys could have improved response rates. However, it does introduce biases that should be acknowledged.

Thus, a chain of evidence was employed to mitigate potential risks and enhance this thesis's observability and transparency. This entails documenting the decisions made at each stage of data selection and analysis, providing a clear rationale for why certain data sources or methods were chosen over others.

6.5.2 Internal Validity

Internal validity regards examining causal relations when different factors could have an impact on the results, looking at what factor has what impact. In this case study, the impact of forking on knowledge loss and code quality is examined. However, it is essential to recognize that certain implications may not have been fully addressed. While additional factors have been considered to some extent (e.g., project size, defects) to provide internal validity, there could be other implications influencing these variables that were beyond this thesis's scope. Therefore the internal validity is threatened.

Another threat to the internal validity is the repository criteria. The repository criteria employed in this thesis played a crucial role in determining which repositories to include in the analysis. This limitation of which repositories to include was applied to projects in case B since they had a polyrepo architecture. Only the repositories that were found in both projects were included. This limitation, while serving the purpose of ensuring consistency and comparability, may have unintentionally introduced bias in the data by excluding certain repositories. By focusing solely on repositories found in both projects, there is a risk of overlooking important variations in the projects. This highlighted the challenge of balancing the need for comparability across the cases while still wanting to avoid biases in the data. Despite the potential bias introduced by this limitation, the decision to use the repository criteria explained in Table 3.3.1 was to achieve comparability in the data. Specifically when analyzing knowledge loss and how it affects code quality since that is the goal of this thesis. This makes it possible for a more direct comparison between the two projects, allowing for a clearer insight into the relationship between knowledge distribution and code quality.

The folder criteria used to determine which folders to include in the analysis are also a threat to internal validity. These criteria were applied to the projects in case A, where the project size differed significantly post-fork. Thus, folder criteria were applied to allow for a more fair comparison between the projects by including folders found in both projects. Similar to the repository criteria this affects the analysis results, and the folder criteria may have unintentionally led to biases in the data. Since the goal of this thesis is to look at knowledge loss and how it affects code quality, the decision to apply these criteria was made to make a clear comparison possible and to help show possible effects on code quality due to knowledge loss.

6.5.3 External Validity

External validity assesses the generalizability of research findings to a broader contexts or cases. In this thesis, external validity was pursued through case selection criteria outlined in chapter 3 to uncover possible trends applicable to similar cases. Nonetheless, it is important to acknowledge the limitations and potential biases inherent in this process. Despite efforts to mitigate biases through careful selection criteria consideration, it is recognized that the findings may not comprehensively represent similar situations, even those with comparable motivations. Furthermore, despite the experience of knowledge loss being a commonality, the findings showed no consistent patterns between the two cases, limiting direct generalization to other contexts. Thus, interpretations of the findings and conclusions should be interpreted with the specific context in which they were studied.

Another potential source of bias regarding case selection is the possibility of overlooking cases that do not align with the established criteria and excluding those that might have been suitable. For instance, the inclusion criterion limited cases to those where both projects had their source code hosted on Github, potentially introducing bias due to convenience sampling and excluding other cases. Additionally, the sample size of this thesis was limited, focusing only on two cases of forking, which may restrict the generalizability of the findings beyond this context. Although efforts were made to ensure diversity, i.e., different motivations for the forks, within the sample, there is a possibility that it has introduced bias.

Furthermore, it is important to recognize that the constructs chosen for this thesis may differ from the definitions used in other research. For example, the different way of calculating knowledge loss using CodeScene than what was used by Izquierdo-Cortazar et al. [43].

6.5.4 Reliability

Reliability serves as the bedrock for ensuring the consistency and replicability of research findings. However, when analyzing the replicability of this thesis, it becomes apparent that certain threats may compromise this reliability.

One concern regarding reliability arises from methodological challenges in semi-structured interviews. While they allow for an in-depth exploration, they pose challenges in maintaining consistency due to their flexible nature. Comparing responses can be difficult since interviewers deviate from predetermined questions, making it challenging to ensure validity and impartiality across participants. While this thesis relied on just two interviews, due to factors such as time limitations and confidentiality agreements, one conducted in person and the other via email, this does not inherently undermine the reliability of our findings. The interviewees serve to validate the findings rather than constitute them. Additionally, in order to strengthen our methodology, all procedures were meticulously documented to enhance the reliability of this thesis.

Chapter 7
Conclusion

This chapter presents the conclusion where the research questions are answered based on the results in chapter 5 and discussion in chapter 6. Followed by a section reflecting on potential future work.

7.1 Conclusion

Regarding **RQ1**, in terms of Knowledge Loss duration, all projects that suffered from knowledge loss, as a result of the fork, failed to reach baseline levels within the two-year analysis period. Implying that the knowledge loss caused by forking persisted for an extended duration, making it challenging to regain previous levels of knowledge. In terms of knowledge loss scope, in Case A, the original project had a small scope (29%-40%) of knowledge loss, while the forked project had a larger scope (79%-90%). In case B, the original project saw a wider scope of knowledge loss (55%-73%), compared to the forked project (42%-50%). The outcome in both cases was likely influenced by the number of switchers, including key contributors. Furthermore, in case A, project size likely played a role in shaping the result.

Concerning **RQ2**, no clear association was found between knowledge loss, caused by forking, and changes in code quality. Project size, contributor activity and turnover, potentially resulting from forking, likely influenced the code quality, with no factor appearing more influential than the others.

With regard to **RQ3**, no clear association existed between forking and bug frequency, specifically in scenarios involving significant knowledge loss. It was found that project size could be a contributing factor. However, code churn, feature management, and premature code releases could also have an indirect impact on bug frequency.

Concerning **RQ4**, the limited number of interviews hindered the acquisition of relevant insights from the cases, thereby restricting understanding on how communities mitigated knowledge loss. Thus, this aspect remains unknown.

7.2 Future Work

Future work could be finding more cases to study. Finding cases that fit the criteria for case selection was difficult. Therefore finding other criteria could be considered when trying to find more cases. Studying more cases could help identify if there are any trends, similarities or differences between different forks. Additionally comparing cases that were forked for the same reason with each other. Further research could explore alternative approaches to repository selection and folder selection. Trying out other criteria that balance the need for comparability with the desire to capture the project's variability. There was an intention to conduct interviews to gain qualitative data. However, only two developers agreed to be interviewed. One thing that would be interesting for future work would be to do more interviews and to look deeper into how the fork is experienced by the developers. Another thing that would be interesting to look into is how the projects are managed. For example, look at onboarding procedures, and how they work on sustaining developers in the community to keep their knowledge, documentation and other ways of knowledge retention and how they communicate within the community to encourage the developers to share their knowledge. Also looking at different development methodologies for example, agile and DevOps, and if the development methodology has any impact on the knowledge distribution, potential knowledge loss and code quality after a fork. As well as seeing if there is a difference between the results depending on what methodology is used.

References

[1] Omolola A Adeoye-Olatunde and Nicole L Olenik. Research and scholarly methods: Semi-structured interviews. *Journal of the american college of clinical pharmacy*, 4(10):1358–1367, 2021.

[2] Shay Banon. Dear search guard users. `https://www.elastic.co/blog/dear-search-guard-users`, September 2019. Accessed: 2024-04-09.

[3] Shay Banon. Amazon: Not ok - why we had to change elastic licensing. `https://www.elastic.co/blog/why-license-change-aws`, 2021. Accessed: 2024-04-09.

[4] Markus Borg, Ilyana Pruvost, Enys Mones, and Adam Tornhill. Increasing, not diminishing: Investigating the returns of highly maintainable code. *ArXiv*, abs/2401.13407, 2024.

[5] John Businge, Ahmed Zerouali, Alexandre Decan, Tom Mens, Serge Demeyer, and Coen De Roover. Variant forks - motivations and impediments. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 867–877, Honolulu, HI, USA, March 2022. IEEE.

[6] Simon Butler, Jonas Gamalielsson, Björn Lundell, Christoffer Brax, Johan Sjöberg, Anders Mattsson, Tomas Gustavsson, Jonas Feist, and Erik Lönroth. On company contributions to community open source software projects. *IEEE Transactions on Software Engineering*, 47(7):1381–1401, 2021.

[7] OpenLogic by Perforce and the Open Source Initiative (OSI). 2023 state of open source report open source usage, market trends, analysis. `https://www.openlogic.com/resources/history-of-open-source-adoption`, 2023. Accessed: 2024-04-10.

[8] Sven; chilling, Andreas; Laumer and Ti Weitzel. Is the source strong with you? a fit perspective to predict sustained participation of floss developers. In *ICIS 2011 Proceedings*, Shanghai, China, December 2011. ICIS.

[9] CodeScene. Code health – how easy is your code to maintain and evolve? https://codescene.io/docs/guides/technical/code-health.html. Accessed: 2024-04-09.

[10] CodeScene. Codescene. https://codescene.com. Accessed: 2024-04-09.

[11] CodeScene. Codescene terminology. https://codescene.io/docs/terminology/codescene-terminology.html. Accessed: 2024-04-09.

[12] CodeScene. Configure teams and developers. https://codescene.ibwave.com/docs/configuration/developers-and-teams.html. Accessed: 2024-04-09.

[13] CodeScene. Delivery effectiveness by organizational trends. https://codescene.io/docs/guides/social/brooks-law.html. Accessed: 2024-04-09.

[14] CodeScene. Hotspots. https://codescene.io/docs/guides/technical/hotspots.html. Accessed: 2024-04-09.

[15] CodeScene. Know the possible biases in the data. https://codescene.io/docs/guides/social/bias.html. Accessed: 2024-04-09.

[16] CodeScene. Knowledge distribution. https://codescene.io/docs/guides/social/knowledge-distribution.html. Accessed: 2024-04-09.

[17] CodeScene. Truck factor algorithm. https://github.com/codescene-research/truck_factor. Accessed: 2024-04-09.

[18] Crunchbase. Elastic. https://github.com/codescene-research/truck_factor. Accessed: 2024-04-09.

[19] Elsie DeBrie and David Goeschel. Open source software licenses: Legal implications and practical guidance. *The Nebraska Lawyer*, pages 7–13, March/April 2016.

[20] Thomas Dohmke. 100 million developers and counting. https://github.blog/2023-01-25-100-million-developers-and-counting/, January 2023. Accessed: 2024-01-25.

[21] Elastic. Elasticsearch. https://www.elastic.co/elasticsearch. Accessed: 2024-02-25.

[22] Elastic. Elasticsearch. https://github.com/elastic/elasticsearch?tab=readme-ov-file. Accessed: 2024-04-10.

[23] Elastic. Faq on 2021 license change. https://www.elastic.co/pricing/faq/licensing. Accessed: 2024-02-25.

[24] Elastic. Faq on elastic license 2.0 (elv2). https://www.elastic.co/licensing/elastic-license/faq. Accessed: 2024-02-25.

[25] Elastic. Past releases. https://www.elastic.co/downloads/past-releases#elasticsearch. Accessed: 2024-02-25.

[26] Elastic. What is opensearch? https://www.elastic.co/elasticsearch/opensearch. Accessed: 2024-02-25.

[27] Elastic. Annual report fiscal year 2023. https://ir.elastic.co/files/doc_downloads/OtherDocuments/2023/AnnualMeeting/Annual-Report-Fiscal-Year-2023.pdf, 2023. Accessed: 2024-04-25.

[28] Elastic. Second quarter fiscal 2024 financial results. https://ir.elastic.co/news/news-details/2023/Elastic-Reports-Second-Quarter-Fiscal-2024-Financial-Results/, 2023. Accessed: 2024-04-25.

[29] Karl Fogel. *Producing Open Source Software: How to Run a Successful Free Software Project*. O'Reilly Media, 2nd edition, January 2017. Accessed: 2024-01-25.

[30] National Center for State Courts. Purposive and convenience sampling. https://www.ncsc.org/consulting-and-research/areas-of-expertise/communications,-civics-and-disinformation/community-engagement/toolkit/purposive-and-convenience-sampling. Accessed: 2024-04-19.

[31] Matthieu Foucault, Marc Palyart, Xavier Blanc, Gail C. Murphy, and Jean-Rémy Falleri. Impact of developer turnover on quality in open-source software. In *Association for Computing Machinery*, New York, NY, USA, 2015.

[32] Free Software Foundation. Gnu affero general public license. https://www.gnu.org/licenses/agpl-3.0.en.html. Accessed: 2024-03-12.

[33] Jonas Gamalielsson and Björn Lundell. Sustainability of open source software communities beyond a fork: How and why has the libreoffice project evolved? *Journal of Systems and Software*, 89:128–145, 2014.

[34] Mathieu Goeminne and Tom Mens. Evidence for the pareto principle in open source software activity. In *the Joint Porceedings of the 1st International workshop on Model Driven Software Maintenance and 5th International Workshop on Software Quality and Maintainability*, pages 74–82. Citeseer, 2011.

[35] Lawrence Hecht and Libby Clark. Survey: Open source programs are a best practice among large companies. https://thenewstack.io/survey-open-source-programs-are-a-best-practice-among-large-companies/. Accessed: 2024-02-26.

[36] Joel Parker Henderson. Monorepo vs. polyrepo. https://github.com/joelparkerhenderson/monorepo-vs-polyrepo. Accessed: 2024-04-15.

[37] Open Source Initiative. The mit license. https://opensource.org/license/mit/. Accessed: 2024-03-20.

[38] Open Source Initiative. Open source initiative. https://opensource.org/. Accessed: 2024-06-04.

[39] Open Source Initiative. Osi approved licenses. https://opensource.org/licenses/. Accessed: 2024-03-20.

[40] Open Source Initiative. The open source definition. https://opensource.org/osd/, February 2023. Accessed: 2024-03-20.

[41] Open Source Initiative. The license review process. https://opensource.org/licenses/review-process/, March 2024. Accessed: 2024-03-20.

[42] io.js. io.js. https://github.com/nodejs/build. Accessed: 2024-06-04.

[43] D. Izquierdo-Cortazar, G. Robles, F. Ortega, and J.M. Gonzalez-Barahona. Using software archaeology to measure knowledge loss in software projects due to developer turnover. In *2009 42nd Hawaii International Conference on System Sciences*, pages 1–10, 2009.

[44] Siyuan Jin, Mianmian Zhang, and Yekai Guo. Software code quality measurement: Implications from metric distributions. *ArXiv*, 6 2023.

[45] Frank Karlitschek. Frank karlitschek. https://karlitschek.de/. Accessed: 2024-03-25.

[46] Frank Karlitschek. Nextcloud. https://karlitschek.de/2016/06/nextcloud/. Accessed: 2024-03-25.

[47] Frank Karlitschek. Big changes: I am leaving owncloud, inc. today. https://karlitschek.de/2016/04/big-changes-i-am-leaving-owncloud-inc-today/, April 2016. Accessed: 2024-03-10.

[48] Karlsson, Andreas. Driving Development Resilience: Analyzing Truck Factors across Proprietary and Open-Source Projects, 2023. Student Paper.

[49] KHTML. Khtml. https://github.com/KDE/khtml. Accessed: 2024-06-04.

[50] Björn Lundell, Jonas Gamalielsson, Stefan Tengblad, Bahram Yousefi, Thomas Fischer, Gert Johansson, Bengt Rodung, Anders Mattsson, Johan Oppmark, Tomas Gustavsson, Jonas Feist, Stefan Landemoo, and Erik Lönroth. Addressing lock-in, interoperability, and long-term maintenance challenges through open source: How can companies strategically use open source? In *IFIP International Conference on Open Source Systems*, pages 80–88, April 2017.

[51] Carl Meadows, Jules Graybill, Kyle Davis, and Mehul Shah. Stepping up for a truly open source elasticsearch. https://aws.amazon.com/blogs/opensource/stepping-up-for-a-truly-open-source-elasticsearch/, January 2021. Accessed: 2024-04-09.

[52] MongoDB. Server side public license (sspl). https://www.mongodb.com/legal/licensing/server-side-public-license, October 2018. Accessed: 2024-04-09.

[53] Mathieu Nassif and Martin P. Robillard. Revisiting turnover-induced knowledge loss in software projects. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 261–272, 2017.

[54] Nextcloud. About us. https://nextcloud.com/about/. Accessed: 2024-03-15.

[55] Nextcloud. Meet the team. https://nextcloud.com/team/. Accessed: 2024-03-15.

[56] Nextcloud. Nextcloud. https://github.com/nextcloud. Accessed: 2024-03-15.

[57] Nextcloud. Nextcloud enterprise faq. https://nextcloud.com/faq/. Accessed: 2024-03-15.

[58] Node.js. Node.js. https://nodejs.org/en. Accessed: 2024-06-04.

[59] Linus Nyman, Tommi Mikkonen, Juho Lindman, and Martin Fougère. Perspectives on code forking and sustainability in open source software. In Imed Hammouda, Björn Lundell, Tommi Mikkonen, and Walt Scacchi, editors, *Open Source Systems: Long-Term Sustainability*, pages 274–279, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[60] Winifred Okong'o and Joshua Rumo Arongo Ndiege. Knowledge sharing in open-source software development communities: a review and synthesis. *VINE Journal of Information and Knowledge Management Systems*, (ahead-of-print), 2023.

[61] OpenSearch. About opensearch. https://opensearch.org/about.html. Accessed: 2024-02-10.

[62] OpenSearch. Opensearch. https://github.com/opensearch-project/OpenSearch. Accessed: 2024-03-20.

[63] OpenSearch. Opensearch partners. https://opensearch.org/partners/. Accessed: 2024-04-11.

[64] ownCloud. About us. https://owncloud.com/about-us/. Accessed: 2024-04-11.

[65] ownCloud. Commercial license. https://owncloud.com/licenses-owncloud-commercial/. Accessed: 2024-04-11.

[66] ownCloud. owncloud, agplv3 and the owncloud commercial license. https://owncloud.com/news/owncloud-agplv3-owncloud-commercial-license/. Accessed: 2024-03-09.

[67] ownCloud. owncloud statement concerning the formation of nextcloud by frank karlitschek. https://owncloud.com/news/owncloud-statement-concerning-formation-nextcloud-frank-karlitschek/. Accessed: 2024-03-09.

[68] L.F. Pitt, R.T. Watson, D. Wynn, G. Zinkhan, and P. Berthon. The penguin's window: Corporate brands from an open-source perspective. *Journal of the Academy of Marketing Science*, 34(2):115–127 – 127, 2006.

[69] Poppler. Poppler. https://poppler.freedesktop.org/. Accessed: 2024-06-04.

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

[71] Gregorio Robles and Jesús M. González-Barahona. A comprehensive study of software forks: Dates, reasons and outcomes. In Imed Hammouda, Björn Lundell, Tommi Mikkonen, and Walt Scacchi, editors, *Open Source Systems: Long-Term Sustainability*, pages 1–14, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

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

[73] Mario Schaarschmidt, Gianfranco Walsh, and Harald F.O. von Kortzfleisch. How do firms influence open source software communities? a framework and empirical analysis of different governance modes. *Information and Organization*, 25(2):99–114, 2015.

[74] Amazon Web Services. What is opensearch? `https://aws.amazon.com/what-is/opensearch/`. Accessed: 2024-04-09.

[75] Khaironi Y. Sharif, Michael English, Nour Ali, Chris Exton, J.J. Collins, and Jim Buckley. An empirically-based characterization and quantification of information seeking through mailing lists during open source developers' software evolution. *Information and Software Technology*, 57:77–94, 2015.

[76] The Apache Software Foundation. Apache license, version 2.0. `https://www.apache.org/licenses/LICENSE-2.0`. Accessed: 2024-03-20.

[77] Adam Tornhill and Markus Borg. Code red: the business impact of code quality - a quantitative study of 39 proprietary production codebases. In *Proceedings of the International Conference on Technical Debt*, TechDebt '22, page 11–20, New York, NY, USA, 2022. Association for Computing Machinery.

[78] WebKit. Webkit. `https://webkit.org/`. Accessed: 2024-06-04.

[79] Laurie Williams and Robert R Kessler. *Pair programming illuminated*, volume 1. Addison-Wesley Professional, 2003.

[80] XFree86. Xfree86. `https://www.xfree86.org/`. Accessed: 2024-06-04.

[81] X.org. X.org. `https://x.org/wiki/`. Accessed: 2024-06-04.

[82] XpdfReader. Xpdfreader. `https://www.xpdfreader.com/`. Accessed: 2024-06-04.

[83] Sihan Xu, Ya Gao, Lingling Fan, Zheli Liu, Yang Liu, and Hua Ji. Lidetector: License incompatibility detection for open source software. *ACM Trans. Softw. Eng. Methodol.*, 32(1), February 2023.

[84] Robert K Yin et al. Design and methods. *Case study research*, 3(9.2):84, 2003.

[85] Shurui Zhou, Bogdan Vasilescu, and Christian Kästner. How has forking changed in the last 20 years? a study of hard forks on github. ICSE '20, page 445–456, New York, NY, USA, 2020. Association for Computing Machinery.

Appendices

Appendix A
Interview Questions

Phase 1 Introduction + reading: "This is completely voluntary participation. You are allowed to opt out of the study at any time, and if you choose to do so, this data will be discarded. You are also allowed to decline to answer any questions. Furthermore, while this interview will not be recorded, your responses will be anonymized and documented".

Phase 2

2:1 Tell us a bit about yourself, what do you do/hobbies/work/interests?

2:2 What made you interested in starting contributing to Open source?

2:3 What made you interested in starting contributing to Open source?

2:3 Have you been active in many OSS projects except this?

2:4 What role do you usually take in OSS projects?

2:5 When did you start contributing to [this]?

2:6 What role did you have in [this] project

2:7 Were you by any chance active in both projects post-fork?

2:8 How did you divide the work?

Phase 3 Let's talk a bit more about the forking of [NAME].

3:1 To your understanding, what was the reason/reasons why the fork happened?

3:2 How was the relationship with the other project [name] after the fork?

3:3 Was there any tension?

3:4 Was there any communication between the projects?

3:5 What was the purpose, and in what way?

3:6 According to you, in what ways did the [name] project get affected by the fork? *e.g., Amount of bugs, knowledge loss about the code?*

3:7 What were the most contributing factors that led to this?

3:8 Did you do anything to prevent it?

3:9 How?

3:10 Did you prepare anything before the fork?

3:11 What/What not?

3:12 In retrospect, what could have been done to prevent the other factors?

3:13 How?

3:14 After the fork, did the project [name] undergo many major changes? *e.g., remove many files, change features, change community rules etc.*

3:15 In terms of knowledge loss, to your understanding, what did the OSS communities do to mitigate the potential loss of knowledge caused by the fork - both yours and the other?

Appendix B
Potential Projects

Table B.1: Assessment of Potential Projects. An 'X' signifies unmet criteria, and an '-' indicates unassessed criteria due to other prerequisites not being met.

Cases	projects	C1	C2	C3	C4	C5	C6	C7	C8	C9
1	MySQL (original)	X	-	-	-	-	-	-	-	-
	MariaDB (Fork)	X	-	-	-	-	-	-	-	-
2	Quagga (original)	-	-	-	-	-	-	-	-	X
	FRRouting (Fork)	-	-	-	-	-	-	-	-	X
3	Terraform (original)	-	-	-	-	-	-	-	-	-
	OpenTofu (Fork)	-	-	-	-	-	-	X	-	-
4	Node.js (original)	-	-	-	-	-	-	-	-	-
	io.js (Fork)	X	-	-	-	-	-	-	-	X
5	Chromium (original)	-	-	-	-	-	-	-	-	-
	Brave (Fork)	-	-	-	-	-	-	-	-	X
6	Joomla (original)	X	-	-	-	-	-	-	-	-
	Mambo (Fork)	X	-	-	-	-	-	-	-	-
7	LibreOffice (original)	X	-	-	-	-	-	-	-	-
	CollaboraOffice (Fork)	X	-	-	-	-	-	-	-	-
8	SugarCRM (original)	X	-	-	-	-	-	-	-	-
	SuiteCRM (Fork)	-	-	-	-	-	-	-	-	-
9	sqlite (original)	-	-	-	-	-	-	X	-	-
	libsql(Fork)	-	-	-	-	-	-	X	-	-

Appendix C
Data Sources

Appendix over the repositories used in this thesis.

C.1 Repositories

Table C.1

Project	Repository	Accessed	Link
Elasticsearch	elasticsearch	2024-02-27	`https://github.com/elastic/elasticsearch`
openSearch	opensearch	2024-02-27	`https://github.com/opensearch-project/OpenSearch`
Nextcloud	Activity	2024-02-27	`https://github.com/nextcloud/activity`
	Android	2024-02-27	`https://github.com/nextcloud/android`
	Android-library	2024-02-27	`https://github.com/nextcloud/android-library`
	Server	2024-02-27	`https://github.com/nextcloud/server`
	files_antivirus	2024-02-27	`https://github.com/nextcloud/files_antivirus`
	files_pdfviewer	2024-02-27	`https://github.com/nextcloud/files_pdfviewer`
	firstrunwizard	2024-02-27	`https://github.com/nextcloud/firstrunwizard`

Continuation of Table ??			
Project	**Repository**	**Accessed**	**Link**
	notes	2024-02-27	https://github.com/nextcloud/notes
	apps	2024-02-27	https://github.com/nextcloud/apps
	updater	2024-02-27	https://github.com/nextcloud/updater
	tasks	2024-02-27	https://github.com/nextcloud/tasks
	templateeditor	2024-02-27	https://github.com/nextcloud/templateeditor
ownCloud	Activity	2024-02-27	https://github.com/owncloud/activity
	Android	2024-02-27	https://github.com/owncloud/android
	Android-library	2024-02-27	https://github.com/owncloud/android-library
	Core	2024-02-xx	https://github.com/owncloud/core
	files_antivirus	2024-02-27	https://github.com/owncloud/files_antivirus
	files_pdfviewer	2024-02-27	https://github.com/owncloud/files_pdfviewer
	firstrunwizard	2024-02-27	https://github.com/owncloud/firstrunwizard
	notes	2024-02-27	https://github.com/owncloud/notes
	apps	2024-02-27	https://github.com/owncloud/apps
	updater	2024-02-27	https://github.com/owncloud/updater
	tasks	2024-02-27	https://github.com/owncloud/tasks
	templateeditor	2024-02-27	https://github.com/owncloud/templateeditor

C.2 Commits

The following table includes all commit IDs from GitHub used.

Table C.2: Table over commits used for this case study for all phases in Elasticsearch, OpenSearch, Nextcloud and ownCloud.

Project / Repository	Phase	Commit ID
Elasticsearch	1	8e5ba9a1e4fe6492839e11afee3fd94fcb10af0a
	2	00a5669766f3ddc62b27301dd5e29af6d19b57bd
	3	b825f6461f1de4b25a72990c6a21945872ba3cba
	4	1d0e50e55624a14003aadc3e03ab73af85f652a0
	5	79689fd8997f8bb9f7345138fe6fbab4262891cb
	6	c4aef6f1a6b450952de62640948d2501c85eecec
	7	b4e6408a98fd5b85e46bd5333cf202bf638bfd9a
	8	011e3bcdb97efc0559d29ad83b7e14d1456acfdc
	9	d841ddc927680e50a6182214f5ff12e989b69463
	10	61000ace86db78e78875a0c9709eb155c623b293
OpenSearch	6	aecc7bd005544d79c6524960a32984926ba6718a
	7	95d47502491ef1efa257275604686665f4e33dfb
	8	54364a5d45ed0a20de42abb53cca8f33cfed88eb
	9	c2388b5c74dd0463bbc6b0bb714b8566eeec3d33
ownCloud / **Activity**	1	d66c01824a505bee54c3f0ab8b397a0d7b1a04bf
	2	b6997110eff7875e50471b26349da7961d8f4d6f
	3	6a36e5f0feb2cebd5f940e37a610d2fd06f9a91d
	4	c0ccc7f07867409992bfac21180273edd8456c48
	5	bf92bab86c487011ea264cd2176e8557cb414c79
	6	5ceef7e0fd0f62b8064b466699609b9ad5ade58d
	7	e9e71f85d49f062c35506e3376c9922082f60cac
	8	9d396a416f8738a13dace2bc8d2ba0d3c6f583eb
	9	0c06e3379ed43d44b615904f8cdac01dce6e7bb5
ownCloud / **files_pdfviewer**	1	13ae3d9878bbc54ca08cbe85f6f0cb269e956676
	2	911407cb1c2051d95dc984d20b514d417ac2ae75
	3	70be665d1da441cb44d54c03fce9cbfc6edf38c4
	4	c871eaab18553f84d0069b755959e45ff6e74aad
	5	f55d9e3f795872c29a2aca6494728a46bab3b559
	6	ed5ced0a939305c8cd32a190eb42eeb5edef94ee
	7	8fee4591f253493a7171cafb55980227c96ab5dd
	8	8fee4591f253493a7171cafb55980227c96ab5dd
	9	360bfc788eef9a3ad94404093ac0f0449758de0f
ownCloud / **android-library**	1	ba42f934fc861f6d5120690edd46ba2c502dd705
	2	7ff0bc0d837edea90b075140f8caba308ce7d378
	3	e48aa9c30447d67b0f8ee2054e1fc826108fc5ec
	4	77c0d785c97233e327cfdeba8365b5ab116934c3
	5	785562038e5b2c9847e8eb5f550065d05fd8137c
	6	67d7217d13af47a6abb40e93a195f1ce1df005fd

Continuation of Table C.2

Project / Repository	Phase	Commit ID
	7	bac38d13efa1b4464ebdc9f972257af44e010865
	8	0e7fd8ee8fb1ad082acba8911412571b206ee1f0
	9	813900d60095d0fe9ad886e94343165652daacfb
ownCloud / **notes**	1	e6a29d445bae1780178e710cbb7559207709555e
	2	91b35f150f62e0d51baa2dac3716e08f7b3b01d5
	3	89c59f8b0ae381f4c9630ca1c9d683c596c3dae8
	4	5ace4c533409cd4d3dad6d294c4ec0d58d78619d
	5	617d45a0ca1c55dbd6903a6d53d7bcb3a418fb95
	6	3945de34ff6e9a7e1edde3db23fbe2c3413b9e17
	7	6b6082a85a7412efb261452de1c7c66ccb89d52b
	8	5fb71443955bd9703838e389cef738b72e00a2a7
	9	e9fdbaa3b931e84da0533320f2ed6ac7ee0ab659
ownCloud / **apps**	1	2048a1f93e44c5a90a82982cd53ace447ef128af
	2	c86e65c67d21e8e2a911222444819b8d325c3b5a
	3	da44fcd7da946582edfff244836ed824b8b9cdec
	4	37330e06b869ae07e859613ebc7cad138167b0f8
	5	018bebd72df34dcd45b363f475367a9504405335
	6	55aa1fa5de650827788fa7656f6b284265e0ebd3
	7	7dbdad39da5f59d9e9a16a40de5001d96b9a3de9
	8	66f6b00f75b0f26735fff69cf3dfa5a04c6211a2
	9	499eddc2b50ae40ee51641a582c6b134f4468036
ownCloud / **tasks**	1	931b2c5a2d55f654f7d81a2f1bf8d5d28be1631f
	2	a3ce44cbf97cd01c3cb56c17bb4c1baf43226309
	3	538fcd00ccae711590b26519ac0aa12b1d7397da
	4	58f74234ea31a1b477d3f503b0c4e66f21637209
	5	398eedcb5634dd7c8d9e0be2033bc3076309ac39
	6	a1c6d560335c25a8a030d05a15e885d6b3b1c52b
	7	893eafb0fe2a2bf7317922181e210f8b893b9b57
	8	0ef18453f820649baa8806bdd24ef1e778d68b62
	9	060fd827c4a9d06b065b212cb11219fd7bedb743
ownCloud / **templateeditor**	1	-
	2	5c602caae0742273cc07572dcb4f615f301f68cd
	3	9da585a004c217b5123d8d9395de56c2c739739d
	4	1ecf1e905cb647cf8ae68be3db3b6519115029c7
	5	7548b704a0b21ecd44ea86e342f85636bcbc987d
	6	789a54f1bbc8530fdbca257ad1faed6633e19403
	7	2ebd5f93a686d5d5a644595401d69234a6218500
	8	456d068b84c4db3e7e0b252e6a961cd7fd7da7c3
	9	bb4bc81bc6745a2c3b788fe50292c5c45f3383dc
ownCloud / **firstrunwizard**	1	c2aa31f70377686aaa3fb53d784fa1c4d0d64c8b
	2	d029afd43ac3c0d1ed3baf6ef9b2c4563936f250
	3	ba46262042464ea25e6cf65e84d12de55e50bf59
	4	700582b5e9ebaab608cd103ad5c0dab3a25bba38
	5	ca3434da7b578450fffb681338eb80826682246f

Continuation of Table C.2

Project / Repository	Phase	Commit ID
	6	70b8c1e413ae2d573eb6f47d9a647c97bbac8d37
	7	fd44b221b9560dcd6eb698b8df25b2fb1a4f2ec8
	8	578583069749621db1b364cc63e7415727d0c0d9
	9	6a76efe4ddf1e36b6c5739bff8ca16f9a7834e0b
ownCloud / **Updater**	1	e21de3fb8bc640496a178c07abb6966c05fbdc85
	2	d2d91b717dd90158ec9b784d0f95eaa4d8807e75
	3	c309644a55505934816c47687867dfb08f8aae82
	4	50458e145a2859bb795b6d7818c1ee49167e8f26
	5	b9da449da176ee43204e30e53ec6705d2acaeef4
	6	ae7e759fead0124d2e5c269e60974556bb4ab7e1
	7	d773f186f608796112ce11427f2d701ad2ea9527
	8	b21be09d576c584c082fce7b6701f0056ca1d5c7
	9	76f347bf87a04e6351501a3b62c083b5337fc846
ownCloud / **files_texteditor**	1	ce4e827a2b0f40409a56bfb424646a538de31cd6
	2	29b180df8fd2c6508fb0d744d8959fe595f562b4
	3	68c5a1c5e76db0ca5cf755c5099ac568318a13d4
	4	6306c79e5a66c8cbf93efa957d89a1f6e791ace2
	5	2d1a964869bed951b81ba42395fa045a5acc2901
	6	821aab18f6d4354a9d5d1eee971831c227ecd6ff
	7	885daa980716f66a273307a951022eb3949a483f
	8	172fb0f8e3a65eff3e86aec33e82d88fbce9e035
	9	b858c159a9bc46bbae1586369beb21e9a0d7dbfe
ownCloud / **Core**	1	89bccf71932c792f6ef8b6f2009131f22415f80b
	2	553ce946d3c999823c5b1f6cfde071ea41a80341
	3	1acdef5e347e368ca99014084e30c9b69f1df181
	4	b00db2c9334874c6062a143de2a6b76b44dca35d
	5	1b5797a4e52f67e4d9ea0fe7e2c20bd14f94ade1
	6	d69abec718e56c44ef1677a6685ac2353a386dcf
	7	1b463b8aaa09f3b2dad36566192eb2fed03e1cc3
	8	4104daa165c32e12f450345a76aa652f7d6ff86f
	9	15423b95c5ab0e669b8a89d6c87c9b9ffeb463e8
ownCloud / **Android**	1	445c3dc9ac925588cb1ea08ee913e0d0638ffa37
	2	30f9abb65708d7060fa4078b55f9ab55f30d199a
	3	78158aa4a35e738c6be565eab951a8f0f96eb65e
	4	04fc2e3cc2ebae11d9532eaf66dda56926638f7f
	5	77a9e623136a19f18517928c7e338d15525180a9
	6	2d503310b3cf55b4f13063eb9d90b2d9c876fd42
	7	e194b27e144f4e89f9b883764a99450369d337a7
	8	8ea417a227ba0de0c59e8facd88f9a10080572f7
	9	8c99156d9a0d43aa7e909b9ac516b41303a70b4d
ownCloud / **files_antivirus**	1	-
	2	880aa3c9b4326c73c4ee7d502f55478d063dad11
	3	5bf2dea2e89475ef7d45923e639e2d2836099aae
	4	ccceb8ca33f3a73a5b1bbd7ca1fd18e1e1fe4e82

Continuation of Table C.2

Project / Repository	Phase	Commit ID
	5	a3c0457b58885aaf73a698130ab987ae25d33569
	6	07ffdfa17d23651758d2f7a1eeb079577445e8e9
	7	1037133decb5b331c4631b25111d0571c433c40a
	8	25e5a33b3e63aa325d0d440dc134331ba619a5a3
	9	8df2298c25f833d662e940ca88b7e73a75dfe6de
Nextcloud / **Activity**	6	e2e1629c66818ce0cdf3b235d069b71804755f86
	7	36f29323973e87fa1fcbcfa852af5fefaec67739
	8	d194108f0959b4bcf2b5954117869e54067dcc68
	9	f2a9f561a1d1790e044adbbf15ff4d36cc641d7a
Nextcloud / **Android**	6	77d0751ded38e0024626738bb464d978f4a8dd06
	7	22044ec663a680c81c3122ec06d320b09f53088d
	8	89f9b34885c4e6d06208dde5d16b7e3a7f8f444e
	9	f7f988bc15c9b35f79d55fab34298a1a4bebbd99
Nextcloud / **Android-Library**	6	06578e6498a30c9ce0ef3ede6354ded98e1a1261
	7	8c20873b55a16d74465c93d62d4646d7563c02e3
	8	e3dd340da1f6c486f65ed75cd01e13d488187727
	9	f92e1728b132dc40d22a101108ec9a67f0e0e5e0
Nextcloud / **Server**	6	8a3b6609694ae286da1ebc5025ec8dc728606931
	7	18a8f3654b0732c0fa6a60905439340676fb8187
	8	3578acb1445addfc186c173c4358eaa363daeda7
	9	80a993ca47cf23d7aa672b9752dc417e1bc4da4c
Nextcloud / **files_antivirus**	6	56dd96256edb64bfaebbe9f2fa9e219dc450becc
	7	9c29782e2b0d18c978f35d56993530fa03c727b2
	8	6b21e930917fa03920dbd0d23954a54d9813db8e
	9	80a993ca47cf23d7aa672b9752dc417e1bc4da4c
Nextcloud / **files_pdfviewer**	6	2a8cde91fb3f3280e26aebf5f2dde68c3073656a
	7	a401e72f8edd00e3b6e2d52a2b4171fef01382dc
	8	e7ff2bf1c557ab0a7a50bf227d8c50742958637a
	9	895f48f671b81dae2d3eaf5ba2a59a70d30c0d37
Nextcloud / **firstrunwizard**	6	4f76c81da7c1e1048c7993f26876c94f59ec4b4d
	7	7aacff18a1e63af4474c263fba4e56fa7b95ef08
	8	b27e803a9bc6d1fd28cf7fe6990a78ba103b1168
	9	b2b204032f722ce946d6fd28f5a751081d9b20cd
Nextcloud / **notes**	6	a8d91de255f9d5e6ca8e2e618f036c57fe48463f
	7	7614e40d9ed5ab8a88d1a7e5afa42b0a6f4a1544
	8	6d4515328d3bfa2d276820f703b1df26356bd56b
	9	feb7cb4b12f928ec3ae9183cd6a9624d1a4ae052
Nextcloud / **apps**	6	e0b8d86969954dbc83b65918fa441e79c149f7b3
	7	9968b1c53d1fad2b42a136786b9fa36b8391ccd0
	8	d93ee922810b008af2da244a5bccecc842544260
	9	d7273a72c0e2680871e6ac6ec0218b1e7bec7550
Nextcloud / **updater**	6	668c9f8a94a321f2809936eac438a843493d8f45
	7	8b3d7d083b039c3cde1330661167e476606f0bdb
	8	6e647cd6f6e006402bea6db3defe34dbad7f2534

Continuation of Table C.2		
Project / Repository	**Phase**	**Commit ID**
	9	6609d9a985d9818bf81c57264800e51f4383ef82
Nextcloud / **tasks**	6	70823c21f00fcbd870502069d3d0d31b0449809e
	7	2d14ad12553e7d871812fcb9bce88249a8ffb5c4
	8	1769f563d420c4f2f4810b116c8705b1d98442ac
	9	5b4e7661799d9843652e0acaf7a0f7dffe7671c7
Nextcloud / **templateeditor**	6	c15f360694b0915a2d7f92a768c9950e290de04f
	7	354f4231ab318d8ea9b503ca82fa3bac478d4996
	8	948f40df38ebf3ef57756ed939018d7c2d586c5e
	9	2befa118aaf3e9d3bcf25934040daa32ff71f932

Appendix D
Configurations

Changes made to the original settings:

- **General**
  - Changed ref to the branch we wanted to analyse
  - Changed Hotspot analysis time span (sliding Window) to .....
- **Analysis Plan**
  - Changed analysis frequency from every week to year
  - Unboxed **Auto-delete old analysis results**
- **Exclusions & Filters**
  - (If suitable) added ***.proto** in **Exclude Files.**
  - Added ****/x-pack/**;**/docs/**;**/build-tools-internal/**;**/build-conventions/**** in **Exclude Content**
- **Ticket ID Mapping**
  - Changed Ticket ID Pattern to **#(\d+)**
- **Team/Developers / Clicked on each project / Edit configuration / Alias Mapping**
  - Merged suggested Alias
- **Team/Developers / Clicked on each project / Edit configuration / Developers**
  - Marked former developers

Appendix E
Hotspot files

Table E.1: Hotspot files after 1 year of the fork for each project

Project	Name	File
ownCloud / Nextcloud	File 1	FileDisplayActivity.java
	File 2	OCFileListFragment.java
	File 3	DrawerActivity.java
	File 4	Server.php
	File 5	util.php
	File 6	WebDav.php
	File 7	ShareActivity.java
	File 8	Manager.php
	File 9	ShareFileFragment.java
Elasticsearch / OpenSearch	File 1	Version.java
	File 2	SnapshotsService.java
	File 3	BlobStoreRepository.java
	File 4	Metadata.java
	File 5	IndexMetadata.java
	File 6	RestoreService.java
	File 7	Node.java
	File 8	AggregatorTestCase.java
	File 9	InternalEngineTests.java
	File 10	IndexShardTests.java
	File 11	InternalTestCluster.java
	File 12	SnapshotResiliencyTests.java
	File 13	OpenSearchIntegTestCase.java / ESIntegTestCase.java
	File 14	MetadataCreateIndexServiceTests.java

Continued on next page

Table E.1 – continued from previous page

Project	Name	File
	File 15	OpenSearchException.java / ElasticsearchException.java
	File 16	MetadataCreateIndexService.java
	File 17	ClusterSettings.java

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2024-05-31

EXAMENSARBETE Case Study on Open Source Forks: Impact on Knowledge Distribution and Code Quality
STUDENTER Anja Pettersson, Nova Svensson
HANDLEDARE Markus Borg (LTH), Joseph Fahey (CodeScene)
EXAMINATOR Alma Orucevic-Alagic (LTH)

Forking för Framgång: Hur påverkas Kodkvalitet och projektkunskap?

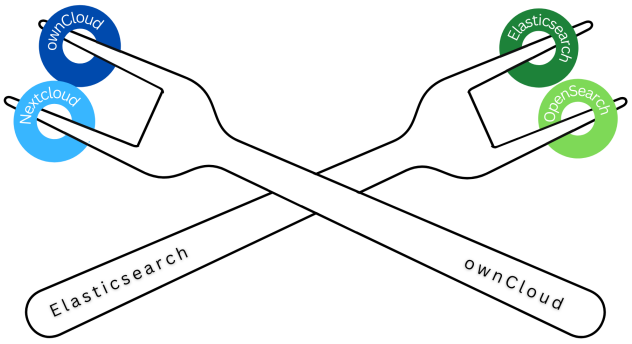
POPULÄRVETENSKAPLIG SAMMANFATTNING **Anja Pettersson, Nova Svensson**

I en tid där mjukvaruutveckling och öppen källkod alltmer vävs samman, väcks en undran om effekterna av förgrening inom dessa projekt. Detta arbete ägnar sig åt att utforska just detta fenomen: hur förgrening av öppen källkod påverkar både kodkvalitet och projektkunskap.

I den konkurrenspräglade miljön för mjukvaruutveckling är "time to market" - den tid det tar för mjukvaruprodukter att nå användarna – avgörande för framgång. Detta har gjort öppen källkod till en alltmer värdefull resurs för industrin genom dess förmåga att möjliggöra snabbare utveckling, men snabbhet ensamt räcker inte. För att säkerställa hållbar framgång krävs även effektiva utvecklingsmetoder, högkvalitativ kod och projektkunskap inom utvecklingsorganisationen.

En vanlig företeelse inom öppen källkod är förgrening, där källkodsprojekt kopieras och utvecklas oberoende av originalversionen. Förgreningen kan sedan antingen integreras tillbaka till ursprungsprojektet eller fortsätta utvecklas separat. Denna företeelse kan främja innovation och minska kostnader, men medför även risker som kan påverka projektets framsteg, såsom minskad projektkunskap till följd av utvecklarförlust. Förståelse av dessa effekter och konsekvenser är därför avgörande för att säkerställa ett projekts långsiktiga framgång.

Detta examensarbete undersöker just detta fenomen, hur förgreningar, som lett till nya projekt, påverkats utifrån projektkunskap och kodkvalitet. Arbetet omfattar två fallstudier av förgreningarna: Elasticsearch och OpenSearch samt ownCloud och Nextcloud.



Resultatet visade inget direkt samband mellan förändrad projektkunskap och kodkvalitet vid förgrening av öppen källkod. Snarare påverkades det av andra faktorer som uppstår vid förgreningen, såsom förändringar i projektets omfattning och antalet erfarna utvecklare. Mer erfarna utvecklare minimerade förlusten av projektkunskap, medan mindre projektstorlek bidrog till bibehållen eller förbättrad kodkvalitet. Framtida forskning bör utforska dessa faktorer närmre för att, tillsammans med detta arbete, skapa riktlinjer och bästa praxis för att hantera potentiella utmaningar. Det kan i sin tur leda till en förbättrad mjukvaruutvecklingsprocess och främja ett mer hållbart och effektivt utvecklande av öppen källkod.