

Understanding the Archived Projects on GitHub

Xiaoya Xia
East China Normal University

Shengyu Zhao
Tongji University

Xinran Zhang
East China Normal University

Zehua Lou
East China Normal University

Wei Wang
East China Normal University

Fenglin Bi
East China Normal University

Abstract—Open source software (OSS) on GitHub is experiencing continued growth and a rapid rise in the creation of projects. As OSS evolved, numerous projects faced decline, with some inevitably degrading into unmaintained status and getting archived by the owners. Understanding the deprecated projects helps deepen the perception of OSS maintenance and evolution. This paper describes the study of 361 popular GitHub projects that have been archived. By reading repository READMEs and sending surveys to project maintainers, we found the software repositories were archived due to being transferred, evolved, or unmaintained. We provide a set of 16 reasons and 10 practices to describe why and how these projects were archived. We further reveal the OSS development activity lifecycle patterns by fitting their commit history curves. We also confirmed the importance of the bus factor risk that would influence the sustainability of open source projects. Archiving a software repository is an explicit indication of repository deprecation. By providing the reason (the why), the strategy (the how), and the lifecycle patterns (the what) of the archived projects, we bring implications and practices that promote the health and sustainability of open source projects and the ecosystem overall.

Index Terms—Open Source Governance, Software Evolution, Software Deprecation

I. INTRODUCTION

Open source has created some of our most widely used technologies, including operating systems, web browsers, and databases, cementing its status as the de facto infrastructure of our digital world [1]. OSS has greatly stimulated technology innovation and efficiency improvement. Together the projects formed an inseparable upstream and downstream ecology [2], while the OSS eco-sustainability has become a key topic. Unlike the traditional software development lifecycle models [3] with clear delivery goals, responsible teams, and milestones, OSS development in the early stage relies highly on self-organized contributors and voluntary work [4], thus also leading to software abandonment.

GitHub, as the most popular social coding platform, greatly influenced the pattern of open source development. According to Octoverse, the official annual report published by the platform, over 70 million developers are proactively creating and contributing to open source on GitHub, and the increasing trend shows acceleration in the post-pandemic era. Modern

open source is shifting towards standardization and commercialization, but we can still observe a considerable amount of large, popular projects that are unmaintained, constituting a ghost town on GitHub [5]. Pre-2017, studies related to unmaintained and failed open source projects [5–7] relied on commit frequency and the keywords in the repository README.md (e.g., “deprecated,” “unmaintained,” etc.) to construct the dataset. GitHub launched the Archive Feature that allows users to explicitly announce the project deprecation and set the repository read-only state. Developers cannot create issues, pull requests, or comments in the project repositories moved to an archived state, but can still fork or star the projects.

Archiving a project repository does not necessarily equal failure but is sometimes a natural terminus and even a transition point during software evolution. Deprecating open source projects is also an important governance strategy to plan for a graceful exit, especially for organizations that seek to protect their reputation [8]. Investigations of archived projects can provide insights as to implications regarding open source project maintenance challenges and practices. To our knowledge, archived projects have not been studied to understand open source project governance, evolution, and lifecycles. To this end, the paper addresses the following research questions:

RQ1: Why are popular open source projects archived?

We carefully selected 361 popular GitHub projects that have been archived, sent out surveys to the maintainers, and manually analyzed the project README.md files. We were able to find text content and answers describing the reasons in 165 projects. A total number of 16 reasons for archiving have been identified and are categorized based on different criteria.

RQ2: How are open source projects deprecated?

We observed 237 projects that appeared to have provided deprecation-related information and adopted some practices to smoothly archive the project. Combined with survey answers, we formulated a three-stage archive strategy with a set of best practices.

RQ3: What activity patterns did the archived projects have?

We observed project curves from three perspectives: the most active development duration, the number of development peaks, and the span of the recession period among the whole development lifecycle.

RQ4: What factors make projects more likely to be sustainable?

Xiaoya Xia and Shengyu Zhao contribute equally.
Wei Wang is the corresponding author.

We combined the evolution metrics defined by the CHAOSS (Community Health Analytics Open Source Software) community [9], fitting a decision tree to observe the most significant factors that classify archived and non-archived repositories.

The main contributions of this research are:

1) We provide the reasons behind these popular projects being archived. Previous studies related to unmaintained open source software [6, 7, 10] did not consider the software migration perspective. Our results fill in the gap and reveal a major reason for software deprecation that is related to open source evolution throughout the whole ecosystem.

2) We provide references for maintainers with a set of practices to smoothly transition project repositories, which are particularly helpful for organizations who maintain a pile of projects to standardize the process. A well-planned and well-performed transition can benefit downstream OSS projects and end-users, as well as the project team itself.

3) We reveal the development activity evolution patterns from full development spans and reinforce the importance of the bus factor risks [11] for a healthy open source project evolution.

Given the sustainability concerns affecting many OSS projects [12–15], our study presents significant insights related to OSS evolution on both the ecosystem and individual levels, providing a reverse perspective to further understand OSS evolution, governance, and sustainability. The project exit mechanism perspective also helps minimize the impact that project deprecation brings to the dependent OSS projects.

We organize the remainder of the paper as follows. Section II presents related work. Section III introduces the dataset. Sections IV–VII present methods and results for each of the four research questions. Section VIII discusses the insights and implications for practitioners and researchers. Section IX presents threats to validity, while Section X concludes the paper.

II. RELATED WORK

We discuss the related work from three perspectives: (1) Open source software maintenance; (2) Open source evolution; and (3) Open source deprecation and failure.

Eghbal [16] discussed the difficulties, risks and challenges of OSS maintenance by interviewing popular open source project maintainers. She proposed that software production should be a one-way mirror, and too much participation on the production side occupies too much of maintainers’ energy. Dias et al. [17] summarized the qualities of great open source maintainers by conducting 33 semi-structured interviews with experienced maintainers and surveys with contributors. They concluded technical excellence and communication are the most recurring attributes. When discussing OSS maintenance, the bus factor is a frequently referred to maintenance risk [10, 18, 19] where projects are highly reliant on a small group of people. Our study also reveals the same dilemma—a project in our dataset perfectly embodies this metaphor: it was deprecated because the main maintainer sadly passed away

and no other developers had the capacity or ability to push it forward.

Lehman et al. [20, 21] have built the largest and best known body of research on the evolution of large, long-lived software systems. Xie et al. [22] attempted to verify Lehman’s laws on OSS evolution. They confirmed several of these laws, with the rest being either confirmed or infirmed depending on the laws’ operational definitions. During the early emergence stage of open source development, researchers also studied how open source projects evolve. Godfrey and Tu [23] studied the evolution of Linux kernel at the system level and within the major subsystems in 2000. Schweik and English [24] conducted a five year empirical study to investigate the factors that lead some open source projects to ongoing collaborative success while many others become abandoned. The study proposed a “conventional theory” of open source software, showing that project start small, and in successful cases, grow slightly larger in terms of team size [12].

To understand why modern open source projects failed, Coelho and Valente [7] studied 104 unmaintained projects and concluded that competition in open source markets is the most common reason for project failure. The study summarized nine reasons that are attributed to project characteristics, development team and the environment, which are also discussed in this paper. In another study [5], the authors also proposed a machine learning model to identify unmaintained projects on GitHub. There are also studies around deprecation of API: Robbes et al. [25] investigated the impact of API deprecation and Nascimento et al. [26] explored how deprecated API evolves in its adoption.

III. DATASET

The rationale of constructing the dataset was by first inspired by the study of Coelho et al. [7] on the reasons of OSS failure, in which they considered the top 5,000 most popular projects on GitHub. We increment the samples from 5,000 to 10,000 most popular public GitHub repositories, using the number of stars as a proxy for popularity. The data was returned by the GitHub API in April 2022. GitHub uses a bool value to label the archive status of a repository and supports a custom search API. For the top 10,000 projects, a total of 406 archived projects were obtained (4.06%). We removed repositories that are not software projects (34 repositories, e.g. ebook, tutorials, and awesome collections); repositories whose native language is not English (11 repositories), ending up with a list of 361 projects.

This study is a systematic analysis of 361 archived projects. To highlight a few examples, we have *angular/angular.js* (59,495 stars), *microsoft/MS – DOS* (16,436 stars), and *rkt/rkt* (8,862 stars). Notably, *angular.js* was archived around the time we extracted the data, making it the most popular project that is archived as of the time of writing. 225 repositories (62%) are owned by organizations and 136 repositories

<https://github.com/railsware/upterm/issues/1301#issue-327003344>

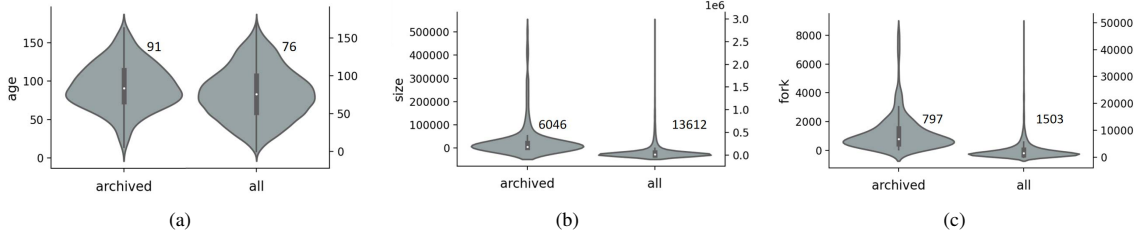


Fig. 1. Distribution of the (a) age, (b) size, (c) forks, without outliers.

(38%) by users. Large corporations and foundations are observed in the form of organizations, such as *facebookarchive* (the organizational account to host Meta’s archived projects, 25 repositories), *microsoft* (7 repositories), *square* (4 repositories), and *apache* (2 repositories). JavaScript is still the unrivaled most popular language (105 repositories, 28%), followed by Java (33 repositories), Python (32 repositories), and Go (30 repositories). In total, the dataset spans 32 programming languages.

Figure 1 shows violin plots with the distribution of age (in months), size (in lines of code), and the number of forks of the 361 archived projects and all 10K projects. The median measures indicate that the archived projects are older and smaller systems with fewer forks. We utilize the Mann-Whitney U test [27] to compare the statistical difference between the two sets of projects. To show the effect size of this difference, we compute Cliff’s delta (d). We found that the effect is median for forks, and small for age and size.

Pre-research: How commonly are projects archived?

During the dataset construction, we investigated how common archived projects occur, aiming to understand the application of this feature as a pre-research. Table I shows the number of total repositories, archived repositories, and the ratio of archived repositories on GitHub, filtered by different star count thresholds. We observed a clear increase in the ratio of archived projects among less popular projects. Among the top 10,000 projects, we also collected 1,068 projects without commits in the last year, among which only 263 projects (25%) were archived. We then limited the sample to the top 5,000 projects and found 439 projects with no commits for over a year and 118 projects (27%) archived, suggesting there are still many unmaintained projects that have not adopted the archiving mechanism.

TABLE I
NUMBER OF PROJECTS ON DIFFERENT STAR THRESHOLDS

stars	10k	5k	2k	1k	500
Archived	66	216	813	1,771	3,611
All	2,347	5,919	16,991	34,306	66,207
Ratio	2.81%	3.65%	4.78%	5.16%	5.45%

IV. RQ1: WHY ARE POPULAR OPEN SOURCE PROJECTS ARCHIVED?

To answer this research question, we inspected project repositories and sent surveys to developers.

A. Methodology

README.md (RM). The README.md file is the very first thing a visitor learns about a project repository. The file often contains rich content describing the project [28], including maintenance status, using the expression similar to “under active development” or “no longer maintained.” For projects that have received a lot of attention (represented by stars), many will provide supplementary information when sunsetting the project and update the information at the beginning of the RM. By walking through 361 RMs, we were able to find texts describing the archive reason for 151 (42%) projects. Some of the texts do not appear directly in the RM but are found in hyperlinks pointing to blog posts, issues, website portals, or social media channels (e.g., Twitter).

Survey. To supplement the data beyond manually collected public information, we sent surveys to project maintainers. The survey was designed with two open-ended questions:

Q1 – Why is this project archived?

Q2 – How did you smoothly deprecate this project?

We were able to find 191 public email addresses. For projects that belong to an organization, we looked for the email address of the principal project contributor. After a period of 20 days, we obtained 34 responses, resulting in a response rate of 17.8%. The survey results supplemented 14 new projects with reasons and revealed new reasons for the projects whose reasons we already had, resulting in a total of 165 projects with reasons. For the sake of convenience and to preserve anonymity, we use P1-P165 to identify them; in some textual examples collected from surveys, we use Project and Developer to substitute the name of a certain project or developer.

We followed the principles of grounded theory [29] to analyze the qualitative data obtained from survey answers and project repositories. Two researchers performed the analysis via open coding and the constant comparison method [30, 31], where emerging codes are compared with existing codes, and then met to discuss their codes and reach a negotiated agreement. At first, we used the 9 reasons discussed in Coelho et al.’s study as seeds [7]; other codes emerged throughout the coding procedure, most notably the reason related to project

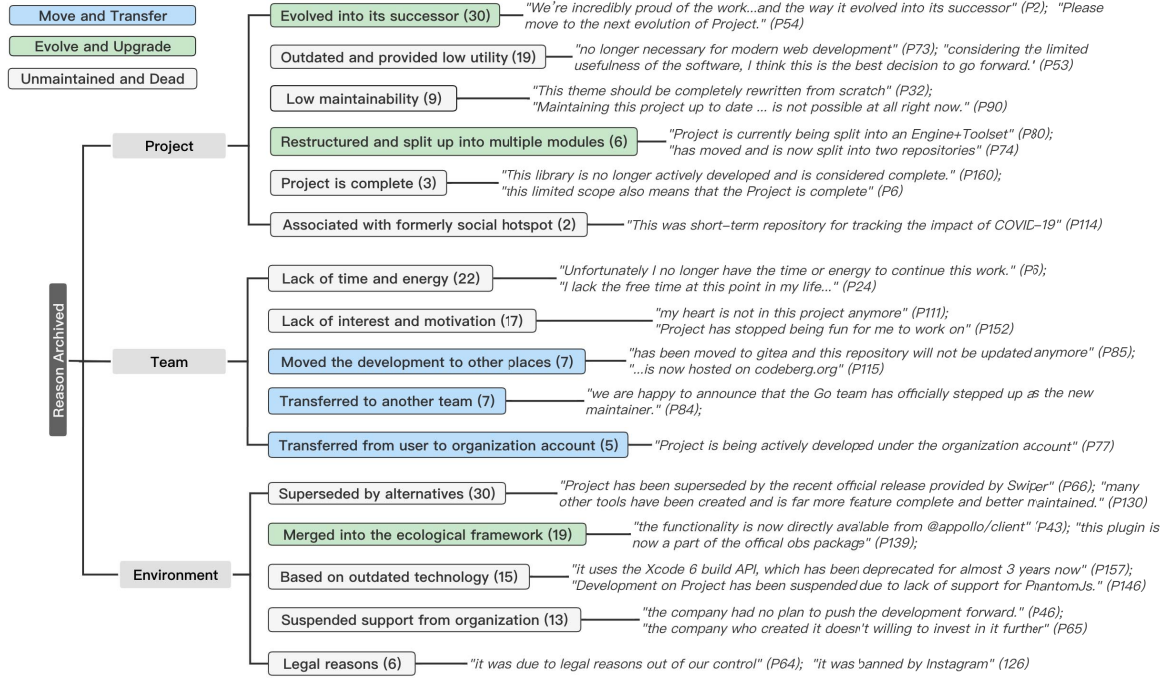


Fig. 2. Reasons and representative examples. The number in parentheses behind each reason represents the number of mentions. A project could report more than one reason.

evolution and migration. We re-worded some of the original codes for a more precise description. For example, we changed “usurped by competitors” to “superseded by alternatives,” because “competitor” sounded rather strong for supportive relationships we found among projects. Many nodded directly at their “competitors” in the RM when deprecating themselves. An example of that would look like:

“There are modern alternatives today, you can use ... instead of this Project.” (P4)

During the coding procedure, we noticed that no new code emerged after the 104th project, which signaled saturation. After completing open coding, the first author performed daily iterations of axial coding [32] during a one-week period to categorize the codes. Eventually, we got 16 codes describing the reasons why project repositories were archived.

B. Reasons

We present the reasons, the number of times reasons were mentioned, and examples of texts in Figure 2. We use three categories to group the reasons: (a) reasons related to project characteristics; (b) reasons related to the development team; and (c) reasons related to the environment where the project and the development team are located.

A project could apply with multiple reasons; for example, P28 stated in a blog post that *“I simply do not have the time nor the motivation to continue working on this Project”*; P52 issued an announcement to say, *“There are a couple reasons,”* one was related to low maintainability: *“It’s rather*

hard to maintain and pick up the work,” with the other reason being related to developers’ interest: *“Eventually, we also lost interest ... and started putting our focus in new areas.”*

The reasons in blue (3 reasons) are related to repository migration and transfer. Projects can migrate to other development platforms or transition to closed source development. Some projects are transferred to another team. For example, Microsoft transferred a Go extension to the Go community, causing the change of the repository address; other projects, maintained by the same group of developers, decided to transfer the project from user to an organization account to *“bring in more people and manage the permission (P77).”*

The reasons in green (3 reasons) indicate the projects have evolved into something else. Evolving into its successor is the most common reason, followed by being merged into an ecological framework (e.g., PyTorch, React, Babel, etc.) Interestingly, rather than being merged into a bigger framework as modules, some projects would be split up into multiple smaller modules mainly in the interest of better maintainability.

The reasons related to **Move and Transfer** and **Evolve and Upgrade** present a flow pattern in the OSS ecosystem. The former relates mainly to the strategic alignment of the team, while the latter relates mainly to the technical vision of the project. Projects in these cases are not actually dead; instead, they were often rewritten, refactored, and restructured to catch up with the state-of-the-art technology or they were rearranged to align with the strategic team vision [33–35].

The reasons in gray (10 reasons) indicate the projects are unmaintained and dead. The most common reason is superseded by alternatives, followed by lack of time and energy (on the part of the developers) and (project being) outdated and provide low utility. We observed some emotional expressions in announcements of projects that are archived related to the team dimension. Maintainers are feeling burnout [36], lacking support, facing competition and legal challenges. One of the developers mentioned:

“Every single day, when I open my email, I pray that I don’t receive yet another Github email. Pull requests, issues, feedback, fixes, etc.”

Some expressed pity and even guilt of having to make the decision, but in the meantime they are so consumed by the support work. The burnout did not seem to come from the project development but the coordination among inbound requests (including code contribution requests) from the community [16].

Summary: Open source projects are archived mostly because the project is unmaintained and dead, but many are also archived due to reasons related to evolution and migration. The two most common reasons are: superseded by alternatives (dead) and evolved into its successor (evolution). Environment is the most crucial factor, followed by project and team.

V. RQ2: HOW ARE OPEN SOURCE PROJECTS DEPRECATED?

By observing the practices of how the once very popular projects were archived, we formulated a set of practices for other projects to plan and prepare for their OSS project exit.

A. Methodology

To answer this research question, we rely on (a) the survey answers of Q2; and (b) the clues that can be observed from project repositories. We first coded the textual documents collected from surveys, using the same open coding [31] method introduced in RQ1. Through this we obtained 6 preliminary codes, of which the most frequently mentioned were: “I added a note to the README.md” (19 answers) and “I suggested a new project as alternatives” (14 answers). Based on these codes, we used a two-round repository investigation to identify useful practices.

In the first round, we checked whether these repositories applied the practices mentioned in survey results. The process was conducted by two researchers by interrogating README.md files, reading the latest five issues and pull requests and confirming their status (whether solved, triaged, or closed with a note). During the inspection, we invited a third researcher to discuss and record the new emerging practices. We filtered the repositories where none of the practices applied, resulting in 237 repositories. In this section we used R1-R237 to identify them.

In the second round, two researchers investigated whether the new practices observed in the first round were applied by each of the 237 projects. All the coding and inspecting processes were executed using a card sorting approach [37].

Continuous comparison and discussion proceeded until consensus was reached [38]. Finally, we sorted all the practices and categorized them using a three-stage framework.

B. Results

Table II shows the three-stage deprecation strategy. Below we explain each practice and present examples.

1) *Stage One—Recognize the crossroad in advance*: There always comes a time when the decision on the future of a project has to be made, whatever the reason. Recognizing these crossroads early enables maintainers to take measures in good time and be prepared, including saving the project from decay.

Be aware of project symptoms – There are several symptoms to observe, including:

a) Observed decrease of project participation and adoption (mentioned in 8 projects): “no major development or community engagement showed up in several months” (R171)

b) The future of the project is frequently discussed (mentioned in 5 projects): “we have received quite a few questions from our users regarding the future of the project” (R13)

c) A marked change from the external environment (mentioned in 3 projects): “Xcode 8 won’t load plug-ins ... looks like the era of Xcode plug-ins is over” (R3)

2) *Stage Two—Communicate with the community*: It is important to be candid with contributors and downstream users. Deprecation should be a decision based on community consensus [39], even for very outdated projects.

Publish project transition announcements – 47 projects have posted announcements on blog posts, social media, or issues and then placed the links in the RMs. Notices in RMs were not counted because they are one-sided and do not convey a sense of “communication” or “conversation.” The announcements usually review the story and history of the project, describe problems and difficulties, explain the reasons for transition, introduce the next step and prospects, and finally acknowledge and thank the community.

Discuss challenges and plans – We found 23 issues opened by project owners to discuss the challenges the project was facing and the plan related with repository transition, 20 of which got long discussion threads. The discussions were related to appreciations (98 relevant comments, “Thank you for the work,” to quote an example), suggestions (32 comments, “Put a note in the README and don’t be too fast with the decision”), supports (8 comments, “I would like to help with the project”), concerns (4 comments, “What can we linux users use instead?”) and also objections (2 comments, “I don’t see the point for this”).

Look for potential help and transfer opportunities – We found links in 5 project RMs redirected to open calls looking for potential help. For example, R132 opened a “Maintainer help wanted” discussion thread and R220 opened a “Looking for maintainers to take over” issue. They have received a fair number of responses expressing interest (12 and 7 responses, respectively); however, the projects were still archived and deprecated “due to lack of follow through” (R220).

TABLE II
THREE-STAGE DEPRECATION STRATEGY

Stage	Practice	Projects
Stage One—Recognize the crossroad in advance	Be aware of project symptoms	#
	a) <i>Observed decrease of project participation and adoption</i>	8
	b) <i>The future of the project is frequently discussed</i>	5
	c) <i>A marked change from the external environment</i>	3
Stage Two—Communicate with the community	Publish project transition announcements	47
	Discuss challenges and plans	23
	Look for potential help and transfer opportunities	5
Stage Three—Work toward a smooth transition	Add notes to the README.md	237
	Provide substitute alternatives and possibly migration guide	82
	Triage the opening issues and PRs to solve, merge, or close	45
	Make the active maintained forks available	22
	Provide contact information	10
	Plan and follow a deprecation timeline	7

3) *Stage Three—Work toward a smooth transition: Add notes to the README.md* – The minimum unit of this practice is to explicitly declare the status of the project in the RM, which is always highlighted at the very top of the document. 237 repositories followed this customary practice by using keywords like “no longer (actively) maintained/developed/supported,” “deprecated/deprecation,” “obsolete,” “unmaintained,” “end of life/project,” “migrated/migration,” “moved,” “discontinued,” “retired” and so on. Many projects provided more information aside from the declaration, which can be seen under other Practices.

Provide substitute alternatives and possibly migration guide – 82 projects suggested substitute projects for users, with 9 of them also providing the links to migration guides, as they were mostly evolved into successors. Two examples are as follows:

“You may want to use the Swarm mode built into the Docker Engine instead, or another orchestration system.” (R35)

“Please refer to the Apollo Client migration guide for more details.” (R161)

Triage the opening issues and PRs to solve, merge, or close them – After excluding projects that had disabled issue functionality, we found 45 archived projects with no issues or pull requests open. The first author read the latest five issues and PRs, noting that 21 projects have interacted with the Issue/PR authors instead of directly closing them (always as a bulk task). Walking through the existing bug reports and bug fixes enables a certain period of software usability [40–42] after the repository is set to read-only, while it also facilitates future forks.

Make the active maintained forks available – We found that forking is highly recommended and encouraged by many project owners to “continue using, maintaining, and improving the project” (R26). 22 projects suggested one or more forks that were “actively maintained” at the time the project was archived, resulting in a total of 27 forks. The first author manually checked each fork. Only 1 fork was also archived. 12 forks (44%) still have commits in the recent six months. We discovered that the suggested forks had obtained quite a lot of attention, with a median measure of the star count of 506 and over half the forks have received thousands of stars.

We also compared the date of the last commit of the forks, finding that these suggested forks have been extended for 29 months on average compared to the original project.

Provide contact information – This practice was found in 10 projects. Since all the communication channels are turned off after the repository is archived, it is important to provide contact information for others to report security issues. For projects that are stable natively, many would still be integrated by downstream users for a long period of time despite being archived [26]. Providing contact information can minimize the impact when important security issues are reported. A contact channel also creates an opportunity for others to help revive the project.

Plan and follow a deprecation timeline – 7 projects curated a deprecation timeline. For example, *helm/charts* uses a timeline which gives the community 9 months to deprecate *Helm2* and migrate to *Helm3*. A timeline doesn’t have to be a sophisticated road map. What really matters is to spare users a transition period to react and slowly replace the usage of (the old version of) the project in their development. For example, the maintainer of R164 mentioned in the survey that they had waited for 3 months to officially archive the repository after announcing the project deprecation.

For comparison purposes, the first author manually checked 321 (439-118) unmaintained and non-archived repositories mentioned in the dataset. A considerable number of non-software repositories (101 repositories) were found, suggesting non-software repositories have a high possibility of being unmaintained and are less likely to adopt the archive function; after excluding them, we only found 32 repositories (14.5%) have notes in the RM explaining their project status. Compared with 64% of archived repositories that have added notes, with many of them also adopting other practices, we considered that applying the archive feature can also stimulate projects to adopt the deprecation practices.

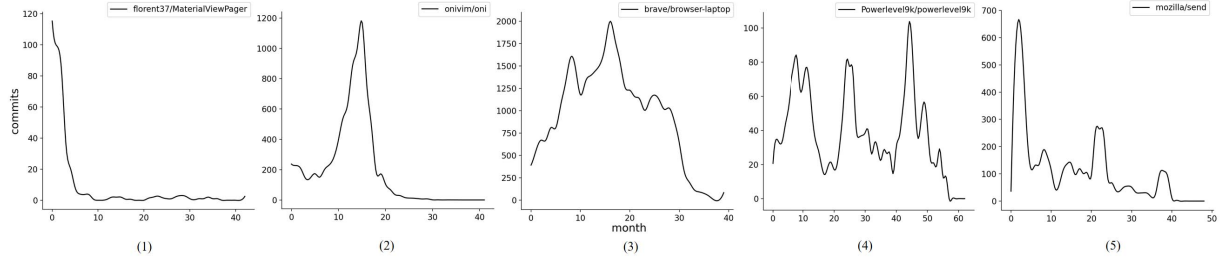


Fig. 3. Representative curves of different activity patterns

Summary: Be aware of project crossroads and communicating with the community are the pre-stages to having a smooth transition of the project and deprecating the repository. Adding notes to the RM is the most adopted practice, but a lot more can be provided in the notes apart from clarifying project status: substitute alternatives, blog post announcements, contact channels, migration guide, deprecation timeline, active forks, etc.

VI. RQ3: WHAT ACTIVITY PATTERNS DID THE ARCHIVED PROJECTS HAVE?

In this section, we discuss different activity lifecycle patterns, aiming to further understand how open source project development evolves.

A. Methodology

We used GHArchive to perform the quantitative analysis in RQ3 and RQ4. GHArchive is an open source project that monitors the public GitHub event timeline and makes it available in hourly archives [43]. The GitHub event data allows us to mine the repository metrics that are related with commits activity, issue activity, pull requests activity, and so on. We only considered projects that were created after 2015/1/1 (186 projects), because the event data was archived with different formats prior to 2015, which would have greatly increased the complexity of the analysis. For these projects, we observed their complete commit number history by monthly aggregation. We define a complete development activity lifecycle (DAL) as the duration period from the project's creation to the date the project was last committed. The median measure of these projects' DAL is 48 months. We used the number of commits as the proxy of repository activity because it represents the unit of code development [44] and was generally used in the previous studies [10, 45, 46]. To better fit the curve characteristics, we used the cubical smoothing algorithm with a three-point approximation [47].

B. Results

Figure 3 presents five representative curves, demonstrating different kinds of patterns. We discuss the three dimensions of the DAL patterns: when were projects most active; how many development peaks appeared; and how long the decline period existed. For simplicity, we used 3 months as the timeframe unit of the observation period.

When were projects most active? We use the timeframe that contains the extreme value of monthly commit count over the whole lifetime to represent the most active development period. 44% of the projects report the period during the first three months (the first timeframe) after its creation, with almost half of them then going straight into a long decline period as shown by the curve in Figure 3.1. For the remaining projects, 24% projects report the peak period in the first half of the project lifecycle (Figures 3.2, 3.3, 3.5) and 32% projects in the second half (Figure 3.4).

How many development peaks appeared? We used the `scipy.signal.find_peaks()` to find the number of development peaks among each curve. Figure 4 (a) shows the distribution of the number of peaks, with most projects showing 2 to 3 peaks. 9% of the projects has only one peak. The maximum was 5 peaks, with 8% of the projects having this many peaks.

How long did the recession last? We define the recession period as the duration from the date the project reported the last development peak to the date it had the last commit, and the active period as the duration before the recession period. Figures 4 (b), (c), (d) shows the box plots of the distribution of active periods (by month), recession periods (by month), and the proportion of the recession period in the whole project lifecycle duration (by percentage). The median measure of the active period, recession period and the recession period percentage is 21 months, 16 months, and 44%, respectively. The results demonstrate a long recession period among the full lifecycle of the projects.

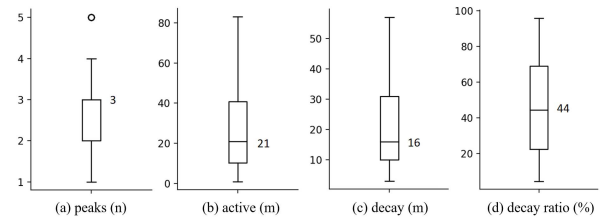


Fig. 4. Distribution for (a) peak number, (b) active months, (c) decay months, and (d) decay ratio

For projects that have only one peak, burstiness is like a flash in the pan. They enjoyed a concentrated development and fell into a long silence, as the representative curves in Figures

3.1 and 3.2 show; Some projects demonstrate an approximate inverted U curve, from which we can observe a continuous development activity with a steady growth and declination process, as shown in Figure 3.3; Some projects demonstrate periodic development cycles, with the peaks highly correlated with project release. Interestingly, in such patterns, some projects held an even activity on each peak (Figure 3.4), while others reflected the recession trend on each peak (Figure 3.5).

Summary: Projects tend to enjoy their vigorous development phase in the early stages (mostly around the time they were created), with 2 to 3 development peaks, and subsequently enter a long recession. The active period usually lasts around 2 years; the recession period usually lasts over a year. The typical activity patterns are: concentratedly active, periodically active, and continuously active.

VII. RQ4: WHAT FACTORS MAKE PROJECTS MORE LIKELY TO BE SUSTAINABLE?

A. Methodology

To answer this research question, we employed a decision tree to examine the characteristics that are important in separating archived projects from non-archived projects. We used machine learning to find useful characteristics, or features, in the data. Because the relationship between the features and the outcome could be nonlinear or the features may interact with one another, we conducted the analysis with decision trees rather than linear or logistic regression models.

The following features were measured for the outcome of archived or non-archived projects. The selection of features referred to the metrics which were defined by the CHAOSS evolution working group. We avoided those metrics that were strongly correlated with the community volume to reduce bias, i.e., the number of commits, number of issues.

- *bus_factor_contributor*: The bus factor is a commonly used metric referring to the number of people you can bear to lose before your project is paralyzed. Here we calculated the smallest number of contributors committing to most development, using 50% as the threshold and the number of commits as the proxy of development workload
- *issue_response_time*: The median measure of the duration between the day an issue was opened to the day it first received comments from accounts that are not issue authors, in days
- *issue_resolution_duration*: The median measure of the duration between the day an issue was opened to the day it was closed for all closed issues, in days
- *pr_merged_duration*: The median measure of the duration between the day a pull request was created to the day it was merged for all the merged pull requests, in days
- *pr_accepted_ratio*: The number of pull requests that are being merged instead of declined/The number of all pull requests that have been opened

We used OpenDigger (<https://github.com/X-lab2017/opendigger>), an open-source data analysis project to extract the metrics data. The positive sample was 186 archived projects that were created after 2015. We then selected the same number of non-archived projects that were also created after 2015 from the initial sample of the top 10K projects. The selection was sorted in descending order of star counts and excluded non-software projects.

B. Results

Figure 5 shows the result of applying the decision tree using scikit-learn. The feature *bus_factor_contributor* is found to be the most important; that is, the number of contributors who undertake more than half of the project development can significantly influence a project's sustainable evolution. Projects with more bus factor contributors have a higher possibility of surviving, while projects with only one bus factor contributor have a high possibility of being archived. We found projects with the feature *pr_accepted_ratio* being over 85% also have a high chance of having been archived. This can be influenced by some archived projects having triaged their opening PRs while the active projects still left many of them open. It also indicates that projects tend to merge PRs instead of decline them before deprecating the repository. The feature *issue_response_time* also has a slight influence on project sustainability, if *bus_factor_contributor* is one and *issue_response_time* is over 8 days then all the projects have been archived (6 projects). Under the condition where *pr_accepted_ratio* is less than 87%, if the feature *bus_factor_contributor* was more than one and less than 12, more than half of the projects have not been archived; if that value was more than 12, then all the projects have not been archived (8 projects). However, if that value was one, meaning only one developer was pushing the project forward, most projects have been archived.

Summary: The bus factor risk is significant for project sustainability. Archived projects tend to have a smaller number of bus factor contributors, a higher PR acceptance rate, and a longer issue response duration.

VIII. DISCUSSION

In this section, we discuss some of the key findings of our research.

A. Important Insights

1) *Open source code evolves and flows in the ecosystem* : A software repository being archived does not mean its code is static. In the open source ecosystem, there is a constant flow and evolution of code as projects are decomposed (Reason: restructured and split up into multiple modules) or merged (Reason: merged into the ecological framework). Projects also experienced continuous refactoring and team alignment, causing the change of repository address from time to time.

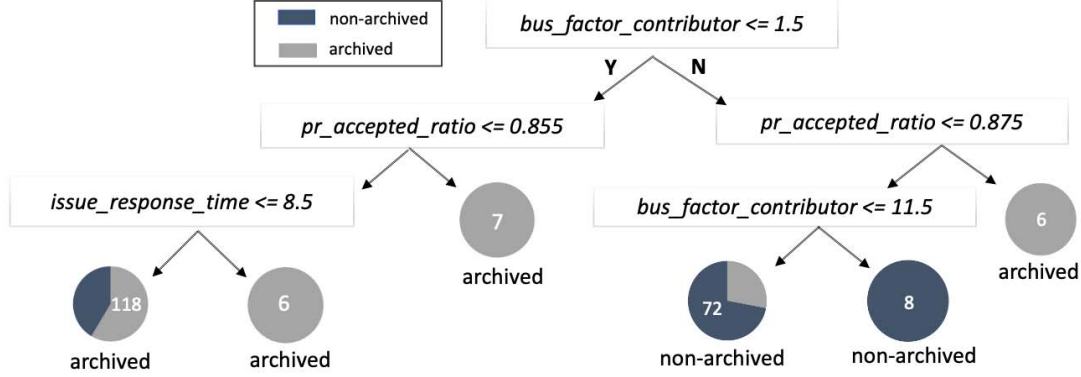


Fig. 5. Decision tree for classifying archived and non-archived projects

Even for those abandoned projects, the code may also live through hard forks.

2) *Maintainers lack effective support*: Maintainer burnout is becoming the subject of interest [16, 36, 48]. With the results of RQ4 reemphasizing the importance of maintainer support (projects with less than 2 bus factor developers are very likely to be archived), we also observed the dilemma and difficulty for maintainers to obtain substantive help. Presumably involving more contributors would alleviate the workload. Indeed, some projects have posted “maintainer help wanted” notices. However, despite many expressed interests, none of them really pushed the project forward. Whatsmore, in some cases, maintainers were actually overwhelmed by opening PRs waiting to contribute, as discussed in Section IV.

3) *Project deprecation requires formalization*: Formalization is noteworthy for organizations who maintain numerous open source projects to mitigate risk and protect their brand reputation. Some companies use an exclusive organizational account to host deprecated repositories, with *facebookarchive*, *googlearchive*, *twitter – archive*, *uber – archive* being notable examples. Simply dropping projects into areas like these is not enough. CNCF has cultivated a better process by designing a deprecation procedure based on TOC review, which was observed through archived projects that belong to CNCF in this research. In Section V, we summarize a set of practices, in which project teams could decide which one(s) to adopt according to each project’s exclusive situation. Still, the results prove that deprecating open source projects is not a one-time decision with a simple click on the “archive” button; clear consideration and appropriate actions are required to minimize the negative influence that deprecation might bring to the community, downstream users, and the OSS ecosystem as a whole.

4) *An informative notice is important*: Whether archived or alive, projects should explicitly describe their maintenance status. When inspecting the RMs, only a few non-archived

repositories with no commit for over a year stated in the RM that the project is “unmaintained” or “active maintained.” It is still not convincing enough for the so-called actively maintained projects, since they have not been committed for a long while, so attaching the date of the notice is also necessary. An archived repository without any explanation will confuse the visitors. The richer the background information, the more it will promote two-way understanding, resulting in convenience for users and the community.

5) *Fork as the mainstream method to resume a dead project*: Fork, as a manner of resuming a project, is frequently mentioned in project deprecation announcements. A maintainer stated the advantage of forking as:

“Forking is nice because it allows the new maintainer to establish their own trust and reputation within the community. It’s also nice because it requires zero involvement from myself.”

A previous study [7] concluded that forks are rarely used on GitHub to continue the failed projects based on the low star count distribution of forks. This was due to taking social forks [49] into consideration. However, our findings prove that hard forks suggested by the original projects have a high possibility of survival and also provide high availability to others (since they received many stars). We infer that being recommended may also significantly contribute to their survival.

6) *Projects tend to be short-term active at creation, long-term inactive before deprecation*: We presumed the activity lifecycle of the archived projects might experience a gradual growth and decline trend. However, most projects show an explosive active trend at the time of their creation, reflecting a weak strength to carry on afterward. This is understandable, since developers are usually most motivated at the outset. The incentives that keep projects robustly developed rather than prematurely exhausted is something that bears further examination.

B. Implications

To organizational open source managers: As mentioned previously, projects being archived does not equal failure. The archive process should also be a well-planned strategy for the

Hard fork refers to copies for creating a divergent version, in contrast; social fork refers to public copies for short-term feature implementation, often with the intention of contributing back

last phase of a project lifecycle. We recommend that open source related organization departments (often referred to as the Open Source Program Office or OSPO) design and facilitate a standardized deprecation process from the top. Our study provides a set of concrete practices for reference. The complete procedure should also be based on each organization’s specific situation and the transition status of a specific project.

To maintainers: We recommend maintainers always stay open and keep in adequate communication with community members, even if a decision has to be made to give up on the project. Our findings show that clear advance communication always receives support and understanding and would benefit downstream users by leaving them space to react. A counterexample is the lack of communication on the part of P135, causing the extended discussion of community members wondering about the team’s roadmap and arguing about the project’s next steps until the team finally declared after one month that:

“Sorry it’s taken so long to respond—it’s often difficult to acknowledge that you can no longer maintain something that you’ve built and nurtured for so many years.”

IX. THREATS TO VALIDITY

Construct validity. The threat comes from the rationality of the questions asked. We are interested in investigating archived projects on GitHub. To achieve this goal, we focus on the application status, reasons, practices, activity history and comparisons with non-archived projects. We believe these questions have high potential to provide unique insights and value for practitioners and researchers.

Non-survivorship bias. As we only investigated archived projects, some findings can only reveal part of the symptoms of the open source world. We found 5 projects had called out for help before being archived, but we don’t know how many were succeed and remain alive. Our findings imply the inclination of actions that developers would take to save the project, which might be helpful for other practitioners.

Dataset bias. When constructing the dataset, we only focus on the popular open source projects on GitHub that have been archived. We believe investigating the reason why projects that received a lot of attention and obtained many downstream users were archived will bring in-depth reevaluation of OSS sustainability. We admit less popular projects might present different characteristics and reasons; for example, there is the Matthew effect [50] that exists in the social coding world [51], where less popular projects might be archived due to lack of attention instead of the maintainer burnout caused by too much attention. We encourage researchers to investigate that perspective. Another threat is that many activities of an open source project take place outside GitHub. To cover more aspects, we inspected all the related hyperlinks provided in the RM, collecting information on project websites and social media. We recognize that such communication channels as Slack and maillist might be more usable but they are hard to scale in this research. We are also aware that project deprecation is associated with more aspects that go beyond

the code repository. For example, the aspects related to how to deal with the licenses, logos and trademarks are not covered but we believe are equally important during project transition. We encourage researchers who specialize in the legal area to look into that as well.

Response bias. As in any survey method, our work can have recall bias—respondents answer only what they recall and not necessarily what was most important to them in the past. Recency and salience can also affect the respondents’ answers. We use open questions, which allow us to collect spontaneous answers, and also combine the answer with information mined from project repository. Indeed, some answers correspond to the information while some answers supplement the information.

Researcher bias. We employed qualitative procedures to classify the textual information. These procedures are subject to interpretation bias. To mitigate the threat, we involved multiple researchers to conduct constant comparison and negotiated agreement. All the researchers have extensive experience both in qualitative methods and OSS.

X. CONCLUSION

In this paper, we consider, from various perspectives, the reasons for archiving projects on GitHub. The most common reasons are related to project evolution and project abandonment. We also reveal a set of practices that popular projects would adopt before setting the repository to read-only. Most projects add at least a note to state the deprecation, but many projects also provide suggestions, guides, contact information and so much more. Furthermore, we show that archived projects are usually most active at the outset and enter quite a long decay period (44% of the whole lifetime, by median measure) at around two to three years after they are created. Finally, we confirm that the bus factor risk can significantly influence project sustainability, whereby open source projects with less than two bus factor contributors face a high possibility of being archived.

As for future work, we propose that researchers further look into the impact of archived projects and validate to what degree each practice discussed in this paper helps mitigate the negative influence. We also encourage researchers and practitioners to amplify the functionality of the Archive Feature—for example, as a clear signal of repository deprecation, “Archive” as a feature could potentially be integrated in the SPDX (Soft Package Data Exchange) and SBOM (Software Bill of Materials) [52, 53] to alert security risks in the supply chain. As we observed many unmaintained projects that are not yet archived, measures should also be taken to enable a wider adoption of this feature. Finally, the ultimate goal of studying archived projects is to better understand OSS sustainability. A healthy and sustainable open source project community is influenced by a complex combination of multiple factors. In the future, we are interested in defining and validating the qualities of healthy OSS communities by means of literature reviews, snowball sampling interviews, quantitative analyses, etc.

XI. ACKNOWLEDGMENT

We would like to thank the developers who took the time to answer our survey. The supplementary data and scripts used in this study can be found online: <https://doi.org/10.5281/zenodo.7230174>

REFERENCES

- [1] InduShobha Chengalur-Smith, Saggi Nevo, and Pindaro Demertzoglou. An empirical analysis of the business value of open source infrastructure technologies. *Journal of the Association for Information Systems*, 11(11):3, 2010.
- [2] Kieran Healy and Alan Schussman. The ecology of open-source software development. Technical report, Technical report, University of Arizona, USA, 2003.
- [3] Nayan B Ruparelia. Software development lifecycle models. *ACM SIGSOFT Software Engineering Notes*, 35(3):8–13, 2010.
- [4] Bo Xu, Donald R Jones, and Bingjia Shao. Volunteers’ involvement in online community based software development. *Information & management*, 46(3):151–158, 2009.
- [5] Jailton Coelho, Marco Tulio Valente, Luciana L Silva, and Emad Shihab. Identifying unmaintained projects in github. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 1–10, 2018.
- [6] Jailton Coelho, Marco Tulio Valente, Luciano Milen, and Luciana L Silva. Is this github project maintained? measuring the level of maintenance activity of open-source projects. *Information and Software Technology*, 122:106274, 2020.
- [7] Jailton Coelho and Marco Tulio Valente. Why modern open source projects fail. In *Proceedings of the 2017 11th Joint meeting on foundations of software engineering*, pages 186–196, 2017.
- [8] [TODO Group]. Shutting down an open source project. <https://todogroup.org/guides/shutting-down/>.
- [9] Sean P Goggins, Matt Germonprez, and Kevin Lombard. Making open source project health transparent. *Computer*, 54(08):104–111, 2021.
- [10] Guilherme Avelino, Eleni Constantinou, Marco Tulio Valente, and Alexander Serebrenik. On the abandonment and survival of open source projects: An empirical investigation. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–12. IEEE, 2019.
- [11] Laurie Williams and Robert R Kessler. *Pair programming illuminated*. Addison-Wesley Professional, 2003.
- [12] Charles M Schweik. Sustainability in open source software commons: Lessons learned from an empirical study of sourceforge projects. *Technology Innovation Management Review*, 3(1), 2013.
- [13] InduShobha Chengalur-Smith, Anna Sidorova, and Sherae L Daniel. Sustainability of free/libre open source projects: A longitudinal study. *Journal of the Association for Information Systems*, 11(11):5, 2010.
- [14] Linus Nyman and Juho Lindman. Code forking, governance, and sustainability in open source software. *Technology Innovation Management Review*, 3(1), 2013.
- [15] Matt Germonprez, Georg JP Link, Kevin Lombard, and Sean Goggins. Eight observations and 24 research questions about open source projects: illuminating new realities. *Proceedings of the ACM on Human-Computer Interaction*, 2(CSCW):1–22, 2018.
- [16] Nadia, Eghbal. *Working in Public: The Making and Maintenance of Open Source Software*. Stripe Press, 2020.
- [17] Edson Dias, Paulo Meirelles, Fernando Castor, Igor Steinmacher, Igor Wiese, and Gustavo Pinto. What makes a great maintainer of open source projects? In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 982–994. IEEE, 2021.
- [18] Guilherme Avelino, Marco Tulio Valente, and Andre Hora. What is the truck factor of popular github applications? a first assessment. *PeerJ PrePrints*, 3:e1233v2, 2015.
- [19] Igor Steinmacher, Gustavo Pinto, Igor Scaliante Wiese, and Marco A Gerosa. Almost there: A study on quasi-contributors in open source software projects. In *Proceedings of the 40th International Conference on Software Engineering*, pages 256–266, 2018.
- [20] Meir M Lehman. Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, 68(9):1060–1076, 1980.
- [21] Meir M Lehman, Juan F Ramil, Paul D Wernick, Dewayne E Perry, and Wladyslaw M Turski. Metrics and laws of software evolution-the nineties view. In *Proceedings Fourth International Software Metrics Symposium*, pages 20–32. IEEE, 1997.
- [22] Guowu Xie, Jianbo Chen, and Iulian Neamtii. Towards a better understanding of software evolution: An empirical study on open source software. In *2009 IEEE international conference on software maintenance*, pages 51–60. IEEE, 2009.
- [23] Qiang Tu et al. Evolution in open source software: A case study. In *Proceedings 2000 International Conference on Software Maintenance*, pages 131–142. IEEE, 2000.
- [24] Charles M Schweik and Robert C English. *Internet success: a study of open-source software commons*. MIT Press, 2012.
- [25] Romain Robbes, Mircea Lungu, and David Röthlisberger. How do developers react to api deprecation? the case of a smalltalk ecosystem. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, pages 1–11, 2012.
- [26] Romulo Nascimento, Andre Hora, and Eduardo Figueiredo. Exploring api deprecation evolution in javascript. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 169–173. IEEE, 2022.

- [27] Nadim Nachar et al. The mann-whitney u: A test for assessing whether two independent samples come from the same distribution. *Tutorials in quantitative Methods for Psychology*, 4(1):13–20, 2008.
- [28] Gede Artha Azriadi Prana, Christoph Treude, Ferdian Thung, Thushari Atapattu, and David Lo. Categorizing the content of github readme files. *Empirical Software Engineering*, 24(3):1296–1327, 2019.
- [29] Anselm Strauss and Juliet M Corbin. *Grounded theory in practice*. Sage, 1997.
- [30] Svetla Georgieva and George Allan. Best practices in project management through a grounded theory lens. *Electronic Journal of Business Research Methods*, 6(1):pp43–52, 2008.
- [31] B Glaser. Theoretical sensitivity: Advances in the methodology of grounded theory procedures and techniques, 1978.
- [32] Juliet Corbin and Anselm Strauss. *Basics of qualitative research: Techniques and procedures for developing grounded theory*. Sage publications, 2014.
- [33] Danilo Silva, Nikolaos Tsantalis, and Marco Tulio Valente. Why we refactor? confessions of github contributors. In *Proceedings of the 2016 24th acm sigsoft international symposium on foundations of software engineering*, pages 858–870, 2016.
- [34] Emerson Murphy-Hill, Chris Parnin, and Andrew P Black. How we refactor, and how we know it. *IEEE Transactions on Software Engineering*, 38(1):5–18, 2011.
- [35] Fabian Fagerholm, Marko Ikonen, Petri Kettunen, Jürgen Münch, Virpi Roto, and Pekka Abrahamsson. Performance alignment work: How software developers experience the continuous adaptation of team performance in lean and agile environments. *Information and Software Technology*, 64:132–147, 2015.
- [36] Naveen Raman, Minxuan Cao, Yulia Tsvetkov, Christian Kästner, and Bogdan Vasilescu. Stress and burnout in open source: Toward finding, understanding, and mitigating unhealthy interactions. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results*, pages 57–60, 2020.
- [37] Jed R Wood and Larry E Wood. Card sorting: current practices and beyond. *Journal of Usability Studies*, 4(1):1–6, 2008.
- [38] Andreas Böhm. Theoretical coding: text analysis in a companion to qualitative research. *A companion to qualitative research*. London: SAGE Publications, pages 270–5, 2004.
- [39] Roy T Fielding. Shared leadership in the apache project. *Communications of the ACM*, 42(4):42–43, 1999.
- [40] Jiaxin Zhu, Minghui Zhou, and Audris Mockus. Effectiveness of code contribution: From patch-based to pull-request-based tools. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 871–882, 2016.
- [41] Akash Balasaheb Dhasade, Akhila Sri Manasa Venigalla, and Sridhar Chimalakonda. Towards prioritizing github issues. In *Proceedings of the 13th innovations in software engineering conference on formerly known as India software engineering conference*, pages 1–5, 2020.
- [42] Tegawendé F Bissyandé, David Lo, Lingxiao Jiang, Laurent Réveillere, Jacques Klein, and Yves Le Traon. Got issues? who cares about it? a large scale investigation of issue trackers from github. In *2013 IEEE 24th international symposium on software reliability engineering (ISSRE)*, pages 188–197. IEEE, 2013.
- [43] [Online]. Gh archive. <http://www.gharchive.org/>.
- [44] Christian Bird, Peter C Rigby, Earl T Barr, David J Hamilton, Daniel M German, and Prem Devanbu. The promises and perils of mining git. In *2009 6th IEEE International Working Conference on Mining Software Repositories*, pages 1–10. IEEE, 2009.
- [45] Qi Xuan and Vladimir Filkov. Building it together: Synchronous development in oss. In *Proceedings of the 36th International Conference on Software Engineering*, pages 222–233, 2014.
- [46] Maëlick Claes, Mika V Mäntylä, Miikka Kuuttila, and Bram Adams. Do programmers work at night or during the weekend? In *Proceedings of the 40th International Conference on Software Engineering*, pages 705–715, 2018.
- [47] Donald L Keefer and Samuel E Bodily. Three-point approximations for continuous random variables. *Management Science*, 29(5):595–609, 1983.
- [48] Danaja Maldeniya, Ceren Budak, Lionel P Robert Jr, and Daniel M Romero. Herding a deluge of good samaritans: How github projects respond to increased attention. In *Proceedings of The Web Conference 2020*, pages 2055–2065, 2020.
- [49] Shurui Zhou, Bogdan Vasilescu, and Christian Kästner. How has forking changed in the last 20 years? a study of hard forks on github. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 445–456. IEEE, 2020.
- [50] Giuliano Bonoli, Bea Cantillon, and Wim Van Lancker. Social investment and the matthew effect. *The uses of social investment*, pages 66–76, 2017.
- [51] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. Social coding in github: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on computer supported cooperative work*, pages 1277–1286, 2012.
- [52] Kate Stewart, Phil Odenec, and Esteban Rockett. Software package data exchange (spdx) specification. *IFOSS L. Rev.*, 2:191, 2010.
- [53] Éamonn Ó Muirí. Framing software component transparency: Establishing a common software bill of material (sbom). 2019.