# Why Modern Open Source Projects Fail

Jailton Coelho, Marco Tulio Valente
Federal University of Minas Gerais
Department of Computer Science
Belo Horizonte, Minas Gerais, Brazil
{jailtoncoelho,mtov}@dcc.ufmg.br

## ABSTRACT

Open source is experiencing a renaissance period, due to the appearance of modern platforms and workflows for developing and maintaining public code. As a result, developers are creating open source software at speeds never seen before. Consequently, these projects are also facing unprecedented mortality rates. To better understand the reasons for the failure of modern open source projects, this paper describes the results of a survey with the maintainers of 104 popular GitHub systems that have been deprecated. We provide a set of nine reasons for the failure of these open source projects. We also show that some maintenance practices—specifically the adoption of contributing guidelines and continuous integration—have an important association with a project failure or success. Finally, we discuss and reveal the principal strategies developers have tried to overcome the failure of the studied projects.

## CCS CONCEPTS

• **Software and its engineering → Risk management**; **Maintaining software**; **Open source model**; *Software evolution*;

## KEYWORDS

Project failure, GitHub, Open Source Software

**ACM Reference format:**
Jailton Coelho, Marco Tulio Valente. 2017. Why Modern Open Source Projects Fail. In *Proceedings of 2017 11th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Paderborn, Germany, September 4–8, 2017 (ESEC/FSE'17),* 11 pages.
https://doi.org/10.1145/3106237.3106246

## 1 INTRODUCTION

Over the years, the open source movement is contributing to a dramatic reduction in the costs of building and deploying software. Today, organizations often rely on open source to support their basic software infrastructures, including operating systems, databases, web servers, etc. Furthermore, most software produced nowadays depends on public source code, which is used for example to encapsulate the implementation of code related to security, authentication, user interfaces, execution on mobile devices, etc. A recent survey shows that 65% out of 1,313 surveyed companies rely on open source to speed application development.[1] For example, Instagram—the popular photo-sharing social network—has a special section of its site to acknowledge the importance of public code to the company.[2] In this page, they thank the open source community for their contributions and explicitly list 25 open source libraries and frameworks used by the social network.

Although open source has its origins in the eighties (or even earlier) [32], the movement is experiencing a renaissance period. One of the main reasons is the appearance of modern platforms and workflows for developing and maintaining open source projects [11]. The most famous example is GitHub; but other platforms are also relevant, such as Bitbucket and GitLab. These platforms modernized the workflow used on open source software development. Instead of changing e-mails with patches, developers contribute to a project by forking it, working and improving the code locally, and then submitting a pull request to the project's leaders.

As a result, developers are creating open source code at a rate never seen before. For example, today GitHub has more than 19 million users and 52 million repositories (without excluding forks). Consequently, these projects are also *failing* at unprecedented rates. Despite this fact, we have very few studies that investigate the *failures faced by open source projects* [1]. We only find similar studies for commercial software. For example, by means of a survey with developers and project managers, Cerpa and Verner study the failure of 70 commercial projects [8]. They report that the most common failures are due to unrealistic delivery dates, underestimated project size, risks not re-assessed through the project, and when staff is not rewarded for working long hours. Certainly, these findings do not apply to open source projects, which are developed without rigid schedules and requirements, by groups of unpaid developers. The Standish Group's CHAOS report is another study frequently mentioned by software practitioners and consultants [34]. The 2007 report mentions that 46% of software projects have cost and schedule problems and that 19% are outright failures. Besides having methodological problems, as pointed by Jørgensen and Moløkken-Østvold [21], this report does not target open source.

This paper describes an investigation with the maintainers of open source projects that have failed, aiming to reveal the reasons for such failures, the maintenance practices that distinguish failed projects from successful ones, the impact of failures on clients, and the strategies tried by maintainers to overcome the failure of their projects. The paper addresses the following research questions:

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*ESEC/FSE'17, September 4–8, 2017, Paderborn, Germany*
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-5105-8/17/09…$15.00
https://doi.org/10.1145/3106237.3106246

[1] https://www.blackducksoftware.com/2016-future-of-open-source
[2] https://www.instagram.com/about/legal/libraries

*RQ1: Why do open source projects fail?* To answer this first RQ we select 542 popular GitHub projects without any commits in the last year. We complemented this selection with 76 systems whose documentation explicitly mentions that the project is abandoned. We asked the developers of these systems to describe the reasons of the projects' failure. Finally, we categorize their responses into nine major reasons.

*RQ2: What is the importance of following a set of best open source maintenance practices?* In this second research question, we check whether the failed projects used a set of best open source maintenance practices, including practices to attract users and to automate maintenance tasks, like continuous integration.

*RQ3: What is the impact of the project failures?* To measure this impact, we counted the number of opened issues and pull requests of the failed projects and also the number of projects that depend on them. The goal is to measure the impact of the studied failures, in terms of affected users, contributors, and client projects.

*RQ4: How do developers try to overcome the projects failure?* In this last research question, we manually analyze the issues of the failed projects to collect strategies and procedures tried by their maintainers to avoid the failures.

We make the following contributions in this paper:

- We provide a list of nine reasons for failures in open source projects. By providing these reasons, using data from real failures, we intend to help developers to assess and control the risks faced by open source projects.
- We reinforce the importance of a set of best open source maintenance practices, by comparing their usage by the failed projects and also by the most and least popular systems in a sample of 5,000 GitHub projects.
- We document three strategies attempted by the maintainers of open source projects to overcome (without success) the failure of their projects.

We organize the remainder of the paper as follows. Section 2 presents the dataset we use to search for failed projects. Section 3 to Section 6 presents answers to each of the four research questions proposed in the study. Section 7 discusses and puts our findings in a wider context. Section 8 presents threats to validity; Section 9 presents related work; and Section 10 concludes the paper.

## 2 DATASET

The dataset used in this paper was created by first considering the top-5,000 most popular projects on GitHub (on September, 2016). We use the number of stars as a proxy for popularity because it reveals how many people manifested interest or appreciation to the project [5]. We limit the study to 5,000 repositories to focus on the maintenance challenges faced by highly popular projects.

We use two strategies to select systems that are no longer under maintenance in this initial list of 5,000 projects. First, we select 628 repositories (13%) without commits in the last year. As examples, we have NVIE/GITFLOW (16,392 stars), MOZILLA/BROWSERQUEST (6,702 stars), and TWITTER/TYPEAND.JS (3,750 stars). Second, we search in the README[3] of the remaining repositories for terms like "deprecated", "unmaintained", "no longer maintained", "no longer supported", and "no longer under development". We found such terms in the READMEs of 207 projects (4%). We then manually inspected these files to assure that the messages indeed denote inactive projects and to remove false positives. After this inspection, we concluded that 76 repositories (37%) are true positives. As an example, we have GOOGLE/GXUI[4] whose README has this comment:

*Unfortunately due to a shortage of hours in a day, GXUI is no longer maintained.*

As an example of false positive, we have TWITTER/LABELLA.JS.[5] In its README, the following message initially led us to suspect that the project is abandoned:

*The API has changed. force.start() and … are deprecated.*

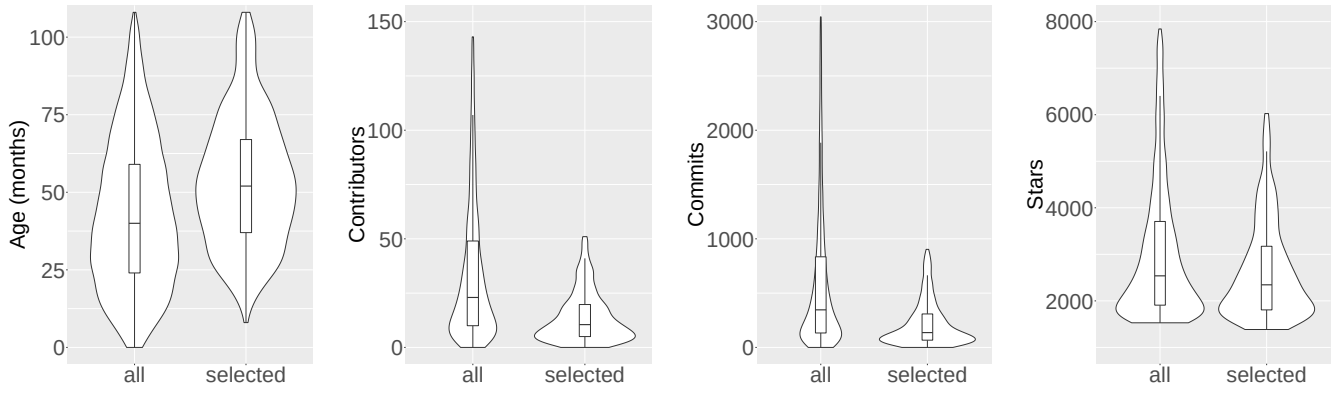
However, in this case, deprecated refers to API elements and not to the project's status. In a final cleaning step, we manually inspected the selected 704 repositores (628 + 76). We removed repositories that are not software projects (51 repositories, e.g., books, tutorials, and awesome lists), repositories whose native language is not English (24 repositories), that were moved to another repository (7 repositories), and that are empty (4 repositories, which received their stars before being cleaned). We ended up with a list of 618 projects (542 projects without commits and 76 projects with an explicit deprecation message in the README).

Figure 1 shows violin plots with the distribution of age (in months), number of contributors, number of commits, and number of stars of the selected repositories. We provide plots for all 5,000 systems (labeled as *all*) and for the 618 systems (12%) considered in this study (labeled as *selected*). The selected systems are older than the top-5,000 systems (52 vs 40 months, median measures); but they have less contributors (11 vs 23), less commits (137 vs 346), and less stars (2,345 vs 2,538). Indeed, the distributions are statistically different, according to the one-tailed variant of the Mann-Whitney U test (p-value $\leq$ 5%). To show the effect size of this difference, we compute Cliff's delta (or *d*). We found that the effect is small for age and commits, medium for contributors, and negligible for stars

GitHub repositories can be owned by a person (e.g., TORVALDS/LINUX) or by an organization (e.g., MOZILLA/PDF.JS). In our dataset, 170 repositories (28%) are owed by organizations and 448 repositories (72%) by users. JavaScript is the most popular language (219 repositories, 36%), followed by Objective-C (98 repositories, 16%), and Java (75 repositories, 12%). In total, the dataset includes systems spanning 26 programming languages. The first paper's author manually classified the application domain of the systems in the dataset, as showed in Table 1. There is a concentration on libraries and frameworks (502 projects, 81%), which essentially reproduces a concentration also happening in the initial list of 5,000 projects.[6]

**Dataset limitations:** The proposed dataset is restricted to popular open source projects on GitHub. We acknowledge that there are

[3] READMEs are the first file a visitor is presented to when visiting a GitHub repository. They include information on what the project does, why the project is useful, and eventually the project status (if it is active or not).
[4] https://github.com/google/gxui
[5] https://github.com/twitter/labella.js
[6] For another research, we classified the domain of the top-5,000 GitHub projects; 59% are libraries and frameworks.




(a) Age (b) Contributors (c) Commits (d) Stars

**Figure 1: Distribution of the (a) age, (b) contributors, (c) commits, and (d) stars, without outliers.**

**Table 1: Application domain of the selected projects**

| Application Domain | Projects |
|---|---|
| Libraries and frameworks | 502 |
| Application software (e.g., text editors) | 63 |
| Software tools (e.g., compilers) | 31 |
| System software (e.g., databases) | 22 |

popular projects in other platforms, like Bitbucket, GitLab or that have their own version control installations. Also, the dataset does not include projects that failed before attracting the attention of developers and users. We consider less important to study such projects, since their failures did not have much impact. Instead, we focus on projects that succeeded to attract attention, users, and contributors, but then failed, possibly impairing other projects.

## 3 WHY DO OPEN SOURCE PROJECTS FAIL?

To answer the first research question, we conducted a survey with the developers of 414 open source projects with evidences of no longer being under maintenance.

### 3.1 Survey Design

The survey questionnaire has three open-ended questions: (1) Why did you stop maintaining the project? (2) Did you receive any funding to maintain the project? (3) Do you have plans to reactivate the project? We avoid asking the developers directly about the reasons for the project failures, because this question can lead to multiple interpretations. For example, an abandoned project could have been an outstanding learning experience to its developers. Therefore, they might not consider that it has failed. In Section 3.3, we detail the criteria we followed to define that a project has failed based on the answers to the survey questions.

Specifically to the developers of the 542 repositories without commits in the last year we added a first survey question, asking them to confirm that the projects are no longer being maintained. We also instructed them to only answer the remaining questions if they agree with this fact. We sent the questionnaire to the repositories' owners or to the project's principal contributor, in the case of repositories owned by organizations. Using this criterion, we were able to find a public e-mail address of 425 developers on GitHub. However, 9 developers are the owners—or the main contributors—of two or more projects. In this case, we only sent one mail to these developers, referring to their first project in number of stars, to avoid a perception of our mails as spam messages.

We sent the questionnaire to 414 developers. After a period of 20 days, we obtained 118 responses and 6 mails returned due to the delivery problems, resulting in a response rate of 29%, which is $118/(414-6)$. To preserve the respondents' anonymity, we use labels D1 to D118 to identify them. Furthermore, when quoting their answers we replace mentions to repositories and owners by *[Project-Name]* and *[Project-Owner]*. This is important because some answers include critical comments about developers or organizations.

Finally, for some projects, we found answers to the first survey question ("Why did you stop maintaining the project?") when inspecting their READMEs. This happened with 36 projects, identified by R1 to R36. As an example, we have the following README:

*Unfortunately, I haven't been able to find the time that I would like to dedicate to this project. (R6)*

Therefore, for the first survey question, we collected 154 answers (118 answers by e-mail and 36 answers from the projects' README). We analyzed these answers using thematic analysis [10, 33], a technique for identifying and recording "themes" (i.e., patterns) in textual documents. Thematic analysis involves the following steps: (1) initial reading of the answers, (2) generating a first code for each answer, (3) searching for themes among the proposed codes, (4) reviewing the themes to find opportunities for merging, and (5) defining and naming the final themes. Steps (1) to (4) were performed independently by each of the paper's authors. After this, a sequence of meetings was held to resolve conflicts and to assign the final themes (step 5).

### 3.2 Survey Results

This section presents the answers to the survey questions. For the 118 developers of systems with no commits in the last year, the

survey included an opening question asking if he/she agrees that the project is no longer under maintenance. 101 developers (86%) confirmed this project condition, as in the following answer:

*Yes, I surely have abandoned the project.* (D20)

By contrast, 17 developers (14%) did not agree with the project status. For example, two developers mentioned work being performed out of the main GitHub repository:

*One current issue that does need to be resolved is that the entire site is served over https, but you wouldn't see that change in the repo.* (D18)

*It is under maintenance. It's just not a lot of people are using it, and I am working on a new breaking version and thus didn't want to commit on the master branch.* (D30)

Next, we present the reasons that emerged after analysing the answers received for the first survey question ("Why did you stop maintaining the project?"). We discuss each reason and give examples of answers associated to them.

**Lack of time:** According to 27 developers, they do not have free time to maintain the projects, as in the following answers:

*It was conceived during extended vacation. When I got back to working I simply didn't have time. Building something like [Project-Name] requires 5-6 hours of work per day.* (D15)

*I was the only maintainer and there was a lot of feature requests and I didn't have enough time.* (D115)

**Lack of interest:** 30 developers answered they lost interest on the projects, including when they started to work on other projects or domains, changed jobs, or were fired.[7] As examples, we have:

*My interest began to wane; I moved to other projects.* (D67)

*I'm not working in the CMS space at the moment.* (D77)

*It became less professionally relevant/interesting.* (D80)

*I was fired by the company that owns the project.* (D65)

**Project is completed:** 17 developers consider that their projects are finished and do not need more features (just few and sporadic bug fixes). As an example, we have the following answers:

*Sometimes, you build something, and sometimes, it's done. Like if you built a building, at some point in time it is finished, it achieved its goals. For [Project-Name] — it achieved all its goals, and it's done. . . . The misconception is that people may mistake an open source project with news. Sometimes there are just no more features to add, no more news — because the project is complete.* (D28)

*I felt it was done. I think the dominant idea is that you have to constantly update every open source project, but in my opinion, this thing works great and needs no updates for any reason, and won't for many, many years, since it's built on extremely stable APIs (namely git and Unix utilities).* (D69)

**Usurped by competitor:** 30 developers answered they abandoned the project because a stronger competitor appeared in the market, as in the case of these projects:

*Google released ActionBarCompat whose goal was the same as [Project-Name] but maintained by them.* (D2)

*The project no longer makes sense. Apple has built technical and legal alternatives which I believe are satisfactory.* (D71)

*It's not been maintained for well over half a year and is formally discontinued. There are better alternatives now, such as SearchView and FloatingSearchView.* (R42)

Specifically, 12 projects explicitly declare in their READMEs that they are no longer maintained due to the appearance of a strong competitor. In all cases, the update date of the project status as unmaintained occurred after appearing the competitor. For example, NODE-JS-LIBS/NODE.IO was declared unmaintained four years after its competitor appeared. We also found this statement in its README: *I wrote node.io when node.js was still in its infancy.*

**Project is obsolete:** According to 21 developers, the projects are not useful anymore, i.e., their features are not more required or applicable.[8] As examples, we have the answers:

*This was only meant as a stopgap to support older OSes. As we dropped that, we didn't need it anymore.* (D11)

*I do not have an app myself anymore using that code.* (D36)

*I personally have no use for it in my work anymore.* (D38)

**Project is based on outdated technologies:** This reason, mentioned by 16 respondents, refer to discontinuation due to outdated, deprecated or suboptimal technologies, including programming languages, APIs, libraries, frameworks, etc. As examples, we have the following answers:

*Due to Apple's abandonment of the Objective-C Garbage Collector which [Project-Name] relied heavily on, future development of [Project-Name] is on an indefinite hiatus.* (R20)

*The core team is now building [Project-Name] in Dart instead of Ruby, and will no longer be maintaining the Ruby implementation unless a maintainer steps up to help.* (R34)

**Low maintainability:** This reason, as indicated by 7 developers, refers to maintainability problems. As examples, we have:

*It is difficult to maintain a browser technology like JavaScript because browsers have very different quirks and implementations.* (D28)

*The project reached an unmaintainable state due to architectural decisions made early in the project's life.* (D30)

**Conflicts among developers:** This reason, indicated by three developers, denotes conflicts among developers or between developers and project owners, as in this answer:

*The project was previously an official plugin—so the [Project-Owner] team worked with me to support it. However, they decided would not longer have the concept of plugins—and they ended the support on their side.* (D73)

The remaining reasons include acquisition by a company, which created a private version of the project (two answers), legal problems (two answers), lack of expertise of the principal developer in the technologies used by the project (one answer), and high demand of users, mostly in the form of trivial and meaningless issues (one answer). Finally, in five cases, it was not possible to infer a clear reason after reading the participant's answers. Thus, we classified

[7] Consequently, these developers do not have more time to work on their projects; however, we reserve the lack of time theme to the cases where the developers still have interest on the projects, but not the required time to maintain them.

[8] The theme does not include projects that are obsolete due to outdated technologies, which have a specific theme.

these cases under an *unclear answer* theme. An example is the following answer: *I am not so sure, but you can probably check the last commit details in GitHub.*

We also asked the participants a second question: *did you receive any funding to maintain the project?* 82 out of 118 answers (69%) were negative. The positive answers mention funding from the company employing the respondent (12 answers), non-profit organizations (three answers; e.g., European Union), and other private companies (two answers). Finally, we asked a third question: *do you have plans to reactivate the project?* Only 18 participants (15%) answered positively to this question.

### 3.3 Combining the Survey Answers

In our study, we consider that a project has *failed* when at least one of the following conditions hold:

(1) The project is no longer under maintenance according to the surveyed developers and they do not have plans to reactivate the project (question #3) and the project is not considered completed (question #1).
(2) The project documentation explicitly mentions that it is deprecated (without considering it completed).

Among the considered answers, 76 projects attend condition (1) and 32 projects attend condition (2). The reasons for the failure of these projects are the ones presented in Section 3.2, except when the themes are *lack of interest* or *lack of time*. For these themes and when the answer comes from the top-developer of a project owned by an organization we made a final check on his number of commits. We only accepted the reasons suggested by developers that are responsible for at least 50% of the projects' commits. For example, D85 answered he stopped maintaining his project due to a lack of time. The project is owned by an organization and D85—although the top-maintainer of the project—is responsible for 30% of the commits. Therefore, in this case, we assumed that it would be possible to other developers to take over the tasks and issues handled by D85. By applying this exclusion criterion, we removed four projects from the list of projects. The final list, which includes reasons for failures according to relevant top-developers or project owners, has 104 projects. In this paper, we call them *failed projects*.

Table 2 presents the reasons for the failure of these projects. The most common reasons are project was *usurped by competitor* (27 projects), project is *obsolete* (20 projects), *lack of time* of the main contributor (18 projects), *lack of interest* of the main contributor (18 projects), and project is based on *outdated technologies* (14 projects). It is also important to note that projects can fail due to multiple reasons, which happened in the case of 6 projects. Thus, the sum of the projects in Table 2 is 110 (and not 104 projects).[9]

As presented in Table 2, we classified the reasons for failures in three groups: (1) reasons related to the development team (including lack of time, lack of interest, and conflicts among developers); (2) reasons related to project characteristics (including project is obsolete, project is based on outdated technologies, and low project maintainability); (3) reasons related to the environment where the project and the development team are placed (including usurpation by competition, acquisition by a company, and legal issues).

[9]The values in Table 2 are not exactly the ones presented in Section 3.2 due to the inclusion and exclusion criteria defined in this section.

**Table 2: Why open source projects fail?**

| Reasons | Group | Projects |
|---|---|---|
| Usurped by competitor | Environment | 27 |
| Obsolete | Project | 20 |
| Lack of time | Team | 18 |
| Lack of interest | Team | 18 |
| Outdated technologies | Project | 14 |
| Low maintainability | Project | 7 |
| Conflicts among developers | Team | 3 |
| Legal problems | Environment | 2 |
| Acquisition | Environment | 1 |

*Summary:* Modern open source projects fail due to reasons related to project characteristics (41 projects; e.g., low maintainability), followed by reasons related to the project team (39 projects; e.g., lack of time or interest of the main contributor); and due to environment reasons (30 projects; e.g., project was usurped by a competidor or legal issues).

## 4 WHAT IS THE IMPORTANCE OF OPEN SOURCE MAINTENANCE PRACTICES?

In this second question, we investigate whether the failed projects followed (or not) a set of best open source maintenance practices, which are recommended when hosting projects on GitHub.[10] Section 4.1 describes the methodology we followed to answer the research question and Section 4.2 presents the results and findings.

### 4.1 Methodology

We analyzed four groups of projects: the 104 projects that have failed, as described in Section 3.3 (*Failed*), the top-104 and the bottom-104 projects by number of stars (*Top* and *Bottom*, respectively), and a random sample of 104 projects (*Random*). *Top*, *Bottom*, and *Random* are selected from the initial sample of top-5,000 projects, described in Section 2, and after applying the same cleaning steps defined in this section. The rationale is to compare the *Failed* projects with the most popular projects in our dataset, which presumably should follow most practices; and also with the least popular projects and with a random sample of projects.

For each project in the aforementioned groups of projects we collected the following information:[11] (1) presence of a README file (which is the landing page of GitHub repositories); (2) presence of a separate file with the project's license; (3) availability of a dedicated site and URL to promote the project, including examples, documentation, list of principal users, etc; (4) use of a continuous integration service (we check whether the projects use Travis CI, which is the most popular CI service on GitHub, used by more than 90% of the projects that enable CI, according to a recent study [18]); (5) presence of a specific file with guidelines for repository contributors; (6) presence of an issue template (to instruct developers to

[10]https://opensource.guide

[11]Five of these maintenance practices are explicitly recommended at: https://help.github.com/articles/helping-people-contribute-to-your-project

**Table 3: Percentage of projects following practices recommended when maintaining GitHub repositories. The effect size reflects the extent of the difference between the repositories in a given group (Top, Bottom, or Random) and the failed projects**

| Maintaince Practice | Failed | Top | Effect | Bottom | Effect | Random | Effect |
|---|---|---|---|---|---|---|---|
| README | 99 | 100 | - | 100 | - | 100 | - |
| License | 61 | 88 | small | 60 | - | 73 | - |
| Home Page | 58 | 87 | small | 52 | - | 60 | - |
| Continuous Integration | 27 | 68 | medium | 41 | - | 45 | small |
| Contributing | 16 | 72 | large | 13 | - | 32 | small |
| Issue Template | 0 | 15 | small | 2 | - | 5 | - |
| Code of Conduct | 0 | 13 | - | 0 | - | 2 | - |
| Pull Request Template | 0 | 3 | - | 0 | - | 0 | - |

write issues according to the repository's guidelines); (7) presence of a specific file with a code of conduct (which is a document that establishes expectations for the behavior of the project's participants [37]); and (8) presence of a pull request template (which is a document to instruct developers to submit pull requests according to the repository's guidelines).

After collecting the data for each project in each group we compared the obtained distributions. First, we analyzed the statistical significance of the difference between the *Top*, *Bottom*, and *Random* groups vs the *Failed* group, by applying the Mann-Whitney test at p-value = 0.05. To show the effect size of the difference, we used Cliff's delta. Following the guidelines of previous work [16, 28, 36], we interpreted the effect size as small for $0.147 < d < 0.33$, medium for $0.33 \leq d < 0.474$, and large for $d \geq 0.474$.

## 4.2 Results

Table 3 shows the percentage of projects following each practice. Despite the group, the most followed practices are the presence of a README file, the presence of a license file, and the availability of a project home page. For example, for the *Failed* projects the percentage of projects following these practices are 99%, 61%, and 58%, respectively. For the *Top* projects, the same values are 100%, 88%, and 87%, respectively. The least followed practices are issue templates, code of conduct, and pull request templates. We did not find a single project following these practices in the *Failed* group. By contrast, 15%, 13%, and 3% of the *Top* projects have these three kind of documents, respectively. In general, we observe the following order among the groups of projects regarding the adoption of the eight considered practices: *Top* > *Random* > *Failed* ≡ *Bottom*. In other words, there is a relevant adoption of most practices by the *Top* projects. By contrast, the 104 projects that failed and that are studied in this paper are more similar to the *Bottom* projects. This fact is reinforced by the analysis of Cliff's delta coefficient. There is a *large* effect size between the adoption of contributing guidelines by the *Top* (72%) and the *Failed* projects (16%), and a *medium* difference in the case of continuous integration services (68% vs 27%). For licenses, home pages, and issue templates, the difference is *small*. For the remaining practices, the difference is negligible or does not exist in statistical terms. In the case of the *Bottom* projects, there is no statistical difference for the eight considered documents. Finally, for *Random*, there is a *small* difference when we consider the use of continuous integration and contributing guidelines.

*Summary:* Regarding the adoption of best open source maintenance practices, the failed projects are more similar to the least popular projects than to the most popular ones. Therefore, these practices seem to have an effect on the success or failure of open source projects. The practices with the most relevant effects are contributing guidelines (large), continuous integration (medium), and licences, home pages, and issue templates (small).

# 5 WHAT IS THE IMPACT OF FAILURES?

With this third research question, we intend to assess the impact of the failure of the studied projects, both to end-users and to the developers of client systems. First, we present the approach we used to answer the question (Section 5.1). Then, we present the results (Section 5.2).

## 5.1 Methodology

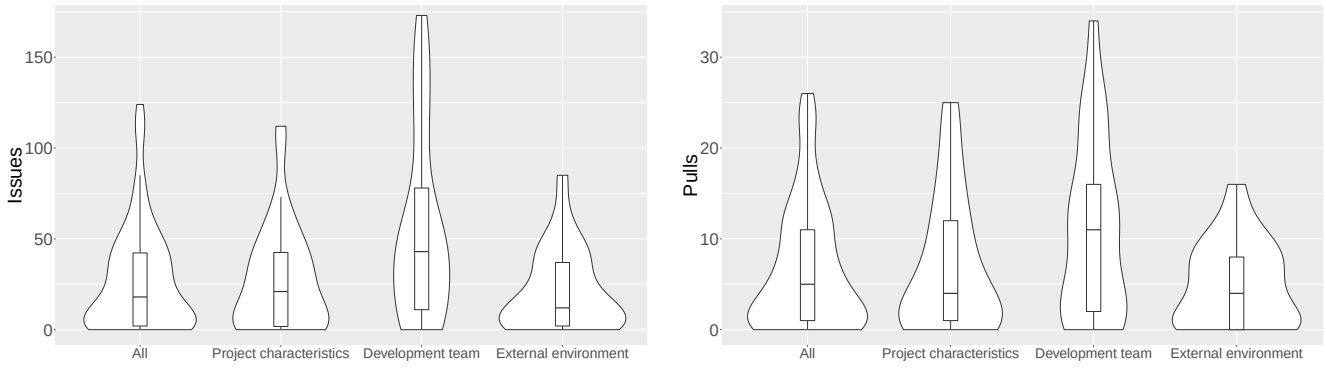
To answer the question, we collected data on (a) the number of issues and pull requests of the failed projects; and (b) the number of systems that depend on these projects, according to data provided by GitHub and by a popular JavaScript package manager.

## 5.2 Results

*Issues and Pull Requests:* Using the GitHub API, we collected the number of opened issues and opened pull requests for each failed project (in the case of issues, we excluded 15 projects that do not use GitHub to handle issues). Our rationale is that one of the negative impacts of an abandoned project is a list of bugs and enhancements (issues) that will not be considered and a list of source code modifications (pull requests) that will not be implemented. Pending issues impact the projects' users, who need to keep using a project with bugs or a frozen set of features, or who will have to migrate to other projects. Pending pull requests contribute to the frustration of the projects' contributors, who will not have their effort appreciated.

*Dependencies:* We also collected data on projects that depend on the failed projects and that therefore are using unmaintained systems. To collect dependencies data, we first rely on a GitHub service that reports the number of client repositories that depend on a given repository.[12] Unfortunately, this feature is available only to Ruby

[12] https://github.com/blog/2300-visualize-your-project-s-community




(a) Opened issues

(b) Opened pull requests

**Figure 2: Distribution of the (a) Opened issues and (b) Opened pull requests, without outliers**

systems. To cover more projects, we also consider dependency data provided by NPM, a popular package manager for JavaScript. As result, we analysed the dependencies of 38 projects, including 10 Ruby projects and 28 JavaScript ones.

*Issues and Pull Requests:* Figure 2 shows violin plots with the distributions of opened issues and pull requests. Considering the 89 failed projects with issues on GitHub, the median number of opened issues is 18 and the median number of pending pull requests is 5. The top-3 failed projects with the highest number of pending issues have 230, 173, and 160 issues. The top-3 failed projects with the highest number of pending pull requests have 54, 45, and 38 pull requests. The figure also shows the number of opened issues and pull requests grouped by failure reasons. The median number of issues for failures associated to project characteristics, development team, and environment reasons are 21, 43, and 12 issues, respectively. For pull requests, the median measures for the same groups of reasons are 4, 11, and 4 pull requests, respectively. By applying Kruskal-Wallis test to compare multiple samples, we find that these distributions are not different.

*Dependencies:* 6 (out of 10) Ruby repositories do not have dependent projects. However, we also found projects with 2,460, 270, 36 and 18 dependents. Regarding the JavaScript systems, 10 (out of 28) projects do not have an entry on NPM (although NPM is very popular, systems can use other package managers or do not use a package manager at all). 15 projects have five or less dependents and three systems have respectively 158, 37, and 13 dependents.

*Summary:* The failed projects have 18 opened issues and 5 opened pull requests (median measures). 55% of the Ruby and JavaScript projects have less than five dependents, which suggests that most clients have also abandoned these projects.

# 6 HOW DO DEVELOPERS TRY TO OVERCOME THE PROJECTS FAILURE?

In this fourth research question, we qualitatively investigate attempts to overcome the failure of the studied projects.

## 6.1 Methodology

The first paper's author read the 20 most recent opened issues and the 20 most recent closed issues of each of the 104 failed projects (in a total of 1,654 issues). As a result, he collected 32 issues where the developers question the status of the projects and/or discuss alternatives to restart the development. The issues, which are identified by I1 to I32, cover 32 projects in the list of failed projects. Examples of titles of selected issues are: *Is this project dead?* (I18), *Is this project maintained?* (I1), and *Is development of this ongoing?* (I7). After this first step, the first author extracted a set of recurrent strategies (or "themes") suggested by developers to overcome the failure of the projects the issues refer to. The proposed themes were validated by the second paper's author, in a last step.

## 6.2 Results

After analyzing the issues, we found three strategies tried by owners or collaborators to overcome the unmaintained status of the projects. Next, we describe these strategies.

**Moving to an organization account:** This strategy, mentioned in five issues, refers to the creation of an organization account with a name similar to the project's name. The hope is that with this kind of account it would be easier to attract new maintainers and to manage permissions. As examples, we have these comments:

*Would creating a [Project-Name] repo in a [Project-Name] org be something people would want?* (I31)

*I am totally cool with setting up an org and transferring control... Just let me know what you need.* (I3)

**Transfer the project to new maintainers:** This strategy, discussed for three projects, consists in a complete transfer of the project's maintainership to other developers (but keeping the project's name), as discussed in these issues:

*Who want to take over this project will be appreciated. We will watch the project together for a while and I will grant every permission.* (I10)

In two projects, a new developer was found and assumed the project, as documented in this issue:

*I have started working on* [Project-Name]. [Project-Owner] *transferred the repository to my account.* (I7)

We tracked the activity of the new maintainers, until February, 2017. They did not perform significant contributions to the projects, despite minor commits. In one project, we found the following complaint about the new maintainer:

*@[Owner-Name] gave this repo to someone who has never been active on GitHub, so this repo is basically dead.* (I11)

**Accepting new core developers:** In five cases, to overcome the low activity on the repositories, volunteers offered to help with the maintenance, as core developers. For example, we have this issue:

*@[Project-Name] Would you be open to adding more collaborators to this repo?* (I17)

In all cases, the proposals were not answered or accepted. As an example, we have this project owner, who requested a detailed maintenance plan before accepting the maintainer:

*I'd be willing to do this if the collaborators provided a roadmap of what they'd like to accomplish with the library.* (I17)

Although it is not exactly an overcome strategy, in 19 cases owners suggested the developers to start collaborating on another project, as in this issue:

*I'd suggest you look at [Project-Name]. It's very active and modern. I'm trying to find time to switch over myself.* (I9)

Finally, although the presented strategies were not able to restart the development of the studied projects, they should not be considered as completely failed ones. To illustrate this fact, we selected 348 projects that almost failed in the year before the study (they have five or less commits). 182 projects (52%) indeed failed in the next year (the studied one). However, 35 projects show evidences of recovering (they have more than the first quartile of commits/year in the studied year, i.e., 15 commits). After inspecting the documentation and issues of these 35 projects, we found that 14 projects attracted new core developers (third strategy), two were transferred to new maintainers (second strategy), and two projects moved to an organization account (first strategy).

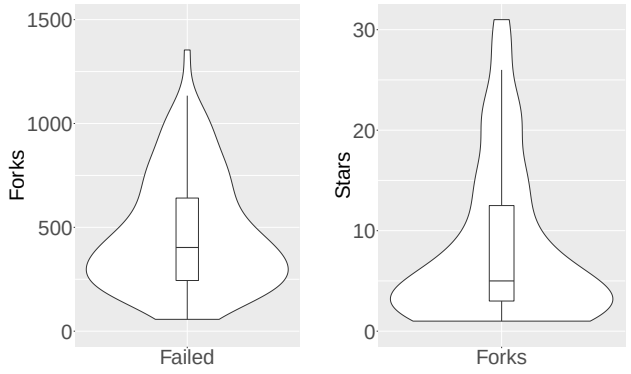
*Summary:* Developers attempted three strategies to overcome the failure of their projects: (a) moving to an organization account; (b) transfer the project to new maintainers; (c) accepting new core developers.

## 6.3 Complementary Investigation: Forks

Forks are used on GitHub to create copies of repositories. The mechanism allows developers to make changes to a project (e.g., fixing bugs or implementing new features) and submit the modified code back to the original repository, by means of pull requests. Alternatively, forks can become independent projects, with their own community of developers. Therefore, forks can be used to overcome the failure of projects, by bootstrapping a new project from the codebase of an abandoned one. For this reason, we decided to complement the investigation of RQ4 with an analysis of the forks of the failed projects.

Figure 3a shows the distribution of the number of forks of the failed projects. They usually have a relevant number of forks, since it is very simple to fork projects on GitHub. The first, median, and third quartile measures are 244, 400, and 638 forks, respectively. The violin plot in Figure 3b aims to reveal the relevance of these forks. For each project, we computed the fork with the highest number of stars. The violin plot shows the distribution of the number of stars of these most successful forks. As we can see, most forks are not popular at all. They are probably only used to submit pull requests or to create a copy of a repository, for backup purposes [20]. For example, the third quartile measure is 13 stars. However, there are two systems with an outlier behavior. The first one is an audio player, whose fork has 1,080 stars. In our survey, the developer of the original project answered that he abandoned the project due to other interests. However, his code was used to fork a new project, whose README acknowledges that this version *"is a substantial rewrite of the fantastic work done in version 1.0 by [Projet-Owner] and others"*. Besides 1,080 stars, the forked project has 70 contributors (as February, 2017). The second outlier is a dependency injector for Android, whose fork was made by Google and has 6,178 stars. The forked project's README mentions that it "is currently in active development, primarily internally at Google, with regular pushes to the open source community".

*Summary:* Forks are rarely used on GitHub to continue the development of open source projects that have failed.




(a) Forks (b) Stars (best fork)

**Figure 3: Distribution of the (a) number of forks of the failed projects and (b) number of stars of the fork with the highest number of stars, for each failed project; both violin plots without outliers**

# 7 DISCUSSION

In this section, we discuss the main findings of our study.

**Completed projects and first Law of Software Evolution:** An interesting finding of the survey with developers is the category of

completed projects (17 systems, 11%), which are considered feature-completed by their developers. They do not plan to evolve these systems, because "*adding more features would just obfuscate the original intent*" (D55) of the projects. Moreover, they also think the projects will not need adaptive maintenance, as in this answer:

*I just stopped working on it because what I have works very well, and will continue working very well until Unix stops being the foundation of most Web development, which basically means until the end of the human race . . . Most projects don't build on similarly solid foundations so they probably need to change more often.* (D56)

Someone can argue that these projects contradict the first Law of Software Evolution, which prescribes that "programs are never completed" [26]. However, Lehman's Laws only apply to E-type systems, where the "E" stands for evolutionary.[13] In these systems, the environment around the program changes and hence the requirements and the program specification [17]. Therefore, Lehman opens the possibility to have completed programs, when they target an environment controlled by the developers or that is very stable (e.g., the Unix ecosystem, as mentioned by Developer D56).

**Competition in open source markets:** The study reveals an important competition between open source projects. The most common reason for project failures is the appearance of a stronger open source competitor (27 projects). Usually, this competitor is the major organization responsible for the ecosystem the project is inserted on, specifically Google (Android ecosystem, 7 projects) and Apple (iOS ecosystem, 5 projects). Therefore, open source developers should be aware of the risks of starting a project that may attract the attention of major players, particularly when the projects have a tight integration and dependency with established platforms, like Android and iOS. Clients should also evaluate the risks of using these "non-official" projects. They should evaluate if it is worth to accept the opportunity costs of delaying the use of a system until it is provided as a built-in service. Alternatively, they can conclude that the costs of delaying the adoption further exceeds the additional benefits of providing earlier a service to end-users. Other competitors mentioned in the survey are D3/D3 (a visualization library for JavaScript) and MVC frameworks, also for JavaScript, such as FACEBOOK/REACT. For example, one developer mentioned that "*high-end front-end development seems to be moving away from jQuery plugins*" (D18). This result confirms that web development is a competitive domain, where the risks of failures are considerable, even for highly popular projects.

**Practical implications:** This study provides insights to the definition of lightweight "maturity models" to open source projects. By lightweight, we mean that such models should be less complex and detailed than equivalent models for commercial software projects, like CMMI [9]. But at least they can prescribe that open source projects should manage and constantly assess the risk factors that emerged from our empirical investigation. We shed light on three particular factors: (a) risks associated to development teams (for example, projects than depend on a small number of core developers may fail due to the lack of time or lack of interest of these developers, after a time working in the project); (b) risks associated to the environment the projects are immersed (which seems to be particularly relevant in the case of projects with a tight integration with mobile operating systems or in the case of web libraries and frameworks); (c) risks associated to project characteristics and decisions, like the use of outdated technologies. Furthermore, we also showed the importance of practices normally recommended to open source development on GitHub. We show that successful projects provide documents like README, contributing guidelines, usage license declarations, and issue templates. They also include a separate home page, to promote the projects among end-users. Finally, we showed evidences on the benefits provided by continuous integration, in terms of automation of tasks like compilation, building, and testing.

[13]The first law (Continuing Changing) is as follows: "An E-type system must be continually adapted, else it becomes progressively less satisfactory in use." [27]

## 8 THREATS TO VALIDITY

The threats to validity of this work are as follows:

**External Validity:** Threats to external validity were partially discussed when presenting the dataset limitations (Section 2). We complement this discussion as follows. First, when investigating the use of continuous integration by the failed, top, bottom, and random projects (RQ2, Section 4), we only consider the use of Travis CI. However, Travis is the most popular CI service on GitHub, used by more than 90% of the repositories that enable CI [18]. Second, the investigation of dependent projects (RQ3, Section 5) only considered systems implemented in Ruby and JavaScript. For JavaScript, we only analyzed dependency data provided by a single package manager system (NPM).

**Internal Validity:** The first threat relates to the selection of the survey participants. We surveyed the project owner, in the case of repositories owned by individuals, or the developer with the highest number of commits, in the case of repositories owned by organizations. Although experts on their projects, it is possible that some participants omitted in their answers the real reasons for the project failures. To mitigate this threat, we avoid asking the participants directly about the causes of the project failures. A second threat relates to the themes denoting reasons for project failures (RQ1) and strategies on how to overcome them (RQ4). We acknowledge that the choice of these themes is to some extent subjective. For example, it is possible that different researchers reach a different set of reasons, than the ones proposed in Section 3.2. To mitigate this threat, the initial selection of themes in RQ1 was performed independently by the two authors of this paper. After this initial proposal, daily meetings were performed during a whole week to refine and improve the initial selection. In the case of RQ4, the themes were proposed by the first paper's author and validated by the second author. A third internal validity threat might appear when interpreting the results of RQ2. In this case, it is important to consider that association does not imply in causation. For example, by just providing contributing guidelines or codes of conduct, a project does not necessarily will succeed.

**Construct Validity:** A first construct validity threat relates to thresholds and parameters used to define the survey sample. We consider as unmaintained the projects that did not have a single commit in the last year (Section 2). We recognize a threat in the selection of this threshold and time frame. However, to mitigate this

threat, we included in the survey 36 projects whose README explicitly declares that the project is unmaintained or deprecated. The second threat concerns the data about the maintenance practices used to answer RQ2 (Section 4). This data was collected automatically, by means of scripts that rely on regular expressions to match different names and extensions used by the documents of interest (e.g., license.md and license.txt). However, we cannot guarantee that the implemented expressions match all possible variations of file names. Moreover, we did not investigate and check the quality of the retrieved documents. For example, we consider that a project has contributing guidelines when this document exists in the repository and it is not empty.

## 9 RELATED WORK

Capiluppi et al. [7] analyze 406 projects from FreshMeat (a deprecated open source repository). For each project, they compute a set of measures along four main dimensions: community of developers, community of users, modularity and documentation, and software evolution. They report that most projects (57%) have one or two developers and that only a few (15%) can be considered active, i.e., continuing improving their popularity and number of users and developers. However, they do not investigate the reasons for the project failures. Khondhu et al. [23] discuss the attributes and characteristics of inactive projects on SourceForge. They report that more than 10,000 projects are inactive (as November, 2012). They also compare the maintainability of inactive projects with other project categories (active and dormant), using the maintainability index (MI) [30]. They conclude that the majority of inactive systems are abandoned with a similar or increased maintainability, in comparison to their initial status. However, there are serious concerns on using MI as a maintainability predictor [3].

Tourani et al. [37] investigate the role, scope and influence of codes of conduct in open source projects. They report that seven codes are used by most projects, usually aiming to provide a safe and inclusive community, as well as dealing with diversity issues. After surveying the literature on empirical studies aiming to validate Lehman's Laws, Fernandez-Ramil et al. [12] report that most works conclude that the first law (Continuing Change) applies to mature open source projects. However, in this work we found 17 completed projects, according to their developers. These projects deal with stable requirements and environments and therefore do not need constant updates or modifications.

Ye and Kishida [38] describe a study to understand what motivates developers to engage in open source development. Using as case study the GIMP project (GNU Image Manipulation Program) they argue that learning is the major driving force that motivates people to get involved in open source projects. However, we do not known if this find applies to the new generation of open source systems, developed using platforms as GitHub. Eghbal [11] reports on the risks and challenges to maintain modern open source projects. She argues that open source plays a key role in the digital infrastructure that sustain our society today. But unlike physical infrastructure, like bridges and roads, open source still lacks a reliable and sustainable source of funding. Avelino et al. concluded that nearly two-thirds of a sample of 133 popular GitHub projects depend on one or two developers to survive [2].

Humphrey [19] presents 12 reasons for project failures, but in the context of commercial software and to justify the adoption of maturity models, like CMMI [9]. The reasons are presented and explained in the form of questions concerning why large software projects are hard to manage, the kinds of management systems needed, and the actions required to implement such systems. Lavallee et al. [24] weekly observed during ten months the development of software projects in a large telecomunnication company. They show that organization factors, e.g., structure and culture, have a major impact on the success or failure of software projects. However, in our study these factors did not appear with the same importance. For example, only three projects failed due to conflicts among developers. We hypothesise this is due to the decentralized and community-centric characteristics of open source code. Washburn et al. [22] analyse 155 postmortems published on the gaming site Gamasutra.com. They report the best practices and common challenges faced in game development and provide a list of factors that impact project outcomes. For example, they found that the creativity of the development team is often a relevant factor in the success or failure of a game. As a practical recommendation, they mention that projects should practice good risk management techniques. We argue that the failure factors elicited in this paper are a start point to include such practices in open source development, i.e., to control the risks we need first to known them.

Recent research on open source has focused on the organization of successful open source projects [29], on how to attract and retain newcomers [6, 25, 31, 35, 39], and on specific features provided by GitHub, such as pull requests [13–15], forks [20], and stars [4, 5].

## 10 CONCLUSION

In this paper, we showed that the top-5 most common reasons for the failure of open source projects are: project was usurped by competitor (27 projects), project became functionally obsolete (20 projects), lack of time of the main contributor (18 projects), lack of interest of the main contributor (18 projects), and project based on outdated technologies (14 projects). We also showed that there is an important difference between the failed projects and the most popular and active projects on GitHub, in terms of following best open source maintenance practices. This difference is more important regarding the availability of contribution guidelines and the use of continuous integration. Furthermore, the failed projects have a non-negligible number of opened issues and pull requests. Finally, we described three strategies attempted by maintainers to overcome the failure of their projects.

As future work, we propose that researchers and practitioners work on defining and validating "maturity models" for open source projects, which can contribute to minimize the risks of adopting these projects in practice. We also recommend investigation on proactive strategies to avoid the failure of projects, for example by identifying and recommending new maintainers with the required expertise to work in projects under threats of being deprecated.

## ACKNOWLEDGMENTS

We would like to thank the 118 GitHub developers who took their time to answer our survey. This research is supported by grants from FAPEMIG, CAPES, and CNPq.

## REFERENCES


[1] Stephanos Androutsellis-Theotokis, Diomidis Spinellis, Maria Kechagia, Georgios Gousios, et al. 2011. Open source software: A survey from 10,000 feet. *Foundations and Trends in Technology, Information and Operations Management* 4, 3–4 (2011), 187–347.

[2] Guilherme Avelino, Leonardo Passos, Andre Hora, and Marco Tulio Valente. 2016. A Novel Approach for Estimating Truck Factors. In *24th International Conference on Program Comprehension (ICPC)*. 1–10.

[3] Dennis Bijlsma, Miguel Alexandre Ferreira, Bart Luijten, and Joost Visser. 2012. Faster issue resolution with higher technical quality of software. *Software Quality Journal* 20, 2 (2012), 265–285.

[4] Hudson Borges, Andre Hora, and Marco Tulio Valente. 2016. Predicting the Popularity of GitHub Repositories. In *12th International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE)*. 1–10.

[5] Hudson Borges, Andre Hora, and Marco Tulio Valente. 2016. Understanding the Factors that Impact the Popularity of GitHub Repositories. In *32nd IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 334–344.

[6] Gerardo Canfora, Massimiliano Di Penta, Rocco Oliveto, and Sebastiano Panichella. 2012. Who is going to mentor newcomers in open source projects?. In *20th International Symposium on the Foundations of Software Engineering (FSE)*. 44–54.

[7] Andrea Capiluppi, Patricia Lago, and Maurizio Morisio. 2003. Characteristics of open source projects. In *7th European Conference on Software Maintenance and Reengineering (CSMR)*. 317–330.

[8] Narciso Cerpa and June M. Verner. 2009. Why did your project fail? *Communications of the ACM* 52, 12 (2009), 130–134.

[9] Mary Beth Chrissis, Mike Konrad, and Sandy Shrum. 2003. *CMMI guidlines for process integration and product improvement.* Addison Wesley.

[10] Daniela S. Cruzes and Tore Dyba. 2011. Recommended steps for thematic synthesis in software engineering. In *5th International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 275–284.

[11] Nadia Eghbal. 2016. *Roads and Bridges: The Unseen Labor Behind Our Digital Infrastructure.* Technical Report. Ford Foundation.

[12] Juan Fernandez-Ramil, Angela Lozano, Michel Wermelinger, and Andrea Capiluppi. 2008. *Empirical Studies of Open Source Evolution.* Springer, 263–288.

[13] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An exploratory study of the pull-based software development model. In *36th International Conference on Software Engineering (ICSE)*. 345–355.

[14] Georgios Gousios, Margaret-Anne Storey, and Alberto Bacchelli. 2016. Work practices and challenges in pull-based development: The contributor's perspective. In *38th International Conference on Software Engineering (ICSE)*. 285–296.

[15] Georgios Gousios, Andy Zaidman, Margaret-Anne Storey, and Arie Van Deursen. 2015. Work practices and challenges in pull-based development: the integrator's perspective. In *37th International Conference on Software Engineering (ICSE)*. 358–368.

[16] Robert J. Grissom and John J. Kim. 2005. *Effect sizes for research: A broad practical approach.* Lawrence Erlbaum.

[17] Israel Herraiz, Daniel Rodriguez, Gregorio Robles, and Jesus M. Gonzalez-Barahona. 2013. The evolution of the laws of software evolution: A discussion based on a systematic literature review. *Comput. Surveys* 46, 2 (2013), 1–28.

[18] Michael Hilton, Timothy Tunnell, Kai Huang, Darko Marinov, and Danny Dig. 2016. Usage, costs, and benefits of continuous integration in open-source projects. In *31st International Conference on Automated Software Engineering (ASE)*. 426–437.

[19] Watts S. Humphrey. 2005. Why big software projects fail: The 12 key questions. *Crosstalk, The Journal of Defense Software Engineering* 3, 18 (2005), 25–29.

[20] Jing Jiang, David Lo, Jiahuan He, Xin Xia, Pavneet Singh Kochhar, and Li Zhang. 2016. Why and how developers fork what from whom in GitHub. *Empirical Software Engineering* 22, 1 (2016), 547–578.

[21] Magne Jørgensen and Kjetil Moløkken-Østvold. 2006. How large are software cost overruns? A review of the 1994 CHAOS report. *Information and Software Technology* 48, 4 (2006), 297–301.

[22] Michael Washburn Jr, Pavithra Sathiyanarayanan, Meiyappan Nagappan, Thomas Zimmermann, and Christian Bird. 2016. What went right and what went wrong: an analysis of 155 postmortems from game development. In *38th International Conference on Software Engineering Companion (ICSE)*. 280–289.

[23] Jymit Khondhu, Andrea Capiluppi, and Klaas-Jan Stol. 2013. Is it all lost? A study of inactive open source projects. In *9th International Conference on Open Source Systems (OSS)*. 61–79.

[24] Mathieu Lavallée and Pierre N. Robillard. 2015. Why good developers write bad code: An observational case study of the impacts of organizational factors on software quality. In *37th International Conference on Software Engineering (ICSE)*. 677–687.

[25] Amanda Lee, Jeffrey C. Carver, and Amiangshu Bosu. 2017. Understanding the Impressions, Motivations, and Barriers of One Time Code Contributors to FLOSS Projects: A Survey. In *39th International Conference on Software Engineering (ICSE)*. 187–197.

[26] Meir M. Lehman. 1980. Programs, life cycles, and laws of software evolution. *IEEE* 68, 9 (1980), 1060–1076.

[27] Meir M. Lehman, Juan F. Ramil, Paul D. Wernick, Dewayne E. Perry, and Wladyslaw M. Turski. 1997. Metrics and laws of software evolution the nineties view. In *4th International Software Metrics Symposium Proceedings (METRICS)*. 20–32.

[28] Mario Linares-Vásquez, Gabriele Bavota, Carlos Bernal-Cárdenas, Massimiliano Di Penta, Rocco Oliveto, and Denys Poshyvanyk. 2013. API change and fault proneness: a threat to the success of Android apps. In *9th International Symposium on the Foundations of Software Engineering (FSE)*. 477–487.

[29] Audris Mockus, Roy T. Fielding, and James D. Herbsleb. 2002. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 11, 3 (2002), 309–346.

[30] Paul Oman and Jack Hagemeister. 1992. Metrics for assessing a software system's maintainability. In *8th International Conference on Software Maintenance (ICSM)*. 337–344.

[31] Gustavo Pinto, Igor Steinmacher, and Marco Aurélio Gerosa. 2016. More common than you think: An in-depth study of casual contributors. In *23th International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. 112–123.

[32] Eric Raymond. 1999. The cathedral and the bazaar. *Knowledge, Technology & Policy* 12, 3 (1999), 23–49.

[33] Danilo Silva, Nikolaos Tsantalis, and Marco Tulio Valente. 2016. Why We Refactor? Confessions of GitHub Contributors. In *24th International Symposium on the Foundations of Software Engineering (FSE)*. 858–870.

[34] Standish Group. 1994. *CHAOS: Project failure and success report report.* Technical Report. MISSING.

[35] Igor Steinmacher, Tayana Uchoa Conte, Christoph Treude, and Marco Aurélio Gerosa. 2016. Overcoming open source project entry barriers with a portal for newcomers. In *38th International Conference on Software Engineering (ICSE)*. 273–284.

[36] Yuan Tian, Meiyappan Nagappan, David Lo, and Ahmed E. Hassan. 2015. What are the characteristics of high-rated apps? a case study on free Android applications. In *31st International Conference on Software Maintenance and Evolution (ICSME)*. 301–310.

[37] Parastou Tourani, Bram Adams, and Alexander Serebrenik. 2017. Code of Conduct in Open Source Projects. In *24th International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. 24–33.

[38] Yunwen Ye and Kouichi Kishida. 2003. Toward an understanding of the motivation Open Source Software developers. In *25th International Conference on Software Engineering (ICSE)*. 419–429.

[39] Minghui Zhou and Audris Mockus. 2015. Who will stay in the FLOSS community? modeling participant's initial behavior. *IEEE Transactions on Software Engineering* 41, 1 (2015), 82–99.