On the abandonment and survival of open source projects: An empirical investigation

Guilherme Avelino*, Eleni Constantinou†, Marco Tulio Valente‡, Alexander Serebrenik§
*Federal University of Piaui, Brazil, gaa@ufpi.edu.br
† University of Mons, Belgium, eleni.constantinou@umons.ac.be
‡ Federal University of Minas Gerais, Brazil, mtov@dcc.ufmg.br
§Eindhoven University of Technology, The Netherlands, a.serebrenik@tue.nl

***Abstract*—*Background*: Evolution of open source projects frequently depends on a small number of core developers. The loss of such core developers might be detrimental for projects and even threaten their entire continuation. However, it is possible that new core developers assume the project maintenance and allow the project to survive. *Aims*: The objective of this paper is to provide empirical evidence on: 1) the frequency of project abandonment and survival, 2) the differences between abandoned and surviving projects, and 3) the motivation and difficulties faced when assuming an abandoned project. *Method*: We adopt a mixed-methods approach to investigate project abandonment and survival. We carefully select 1,932 popular GitHub projects and recover the abandoned and surviving projects, and conduct a survey with developers that have been instrumental in the survival of the projects. *Results*: We found that 315 projects (16%) were abandoned and 128 of these projects (41%) survived because of new core developers who assumed the project development. The survey indicates that (i) in most cases the new maintainers were aware of the project abandonment risks when they started to contribute; (ii) their own usage of the systems is the main motivation to contribute to such projects; (iii) human and social factors played a key role when making these contributions; and (iv) lack of time and the difficulty to obtain push access to the repositories are the main barriers faced by them. *Conclusions*: Project abandonment is a reality even in large open source projects and our work enables a better understanding of such risks, as well as highlights ways in avoiding them.**

***Index Terms*—Project abandonment, Truck factor, Bus factor, Open source development, Core developers**

I. Introduction

Open source software (OSS) is crucial for society. Many proprietary software systems nowadays depend on open source frameworks and libraries, e.g., Instagram publicly acknowledges the developers responsible for the open source libraries used in their site[1]. Moreover, 72% of GitHub survey participants report that they always seek out OSS options when looking for tools[2]. Importance of OSS also implies growing demands on sustainability of OSS projects. Sustainability of OSS projects is, however, a matter of concern since OSS projects are often managed by a small number of developers, without financial support [1]. For example, OpenSSL, a cryptography library used by two-thirds of all Web servers, was maintained by a single developer until 2014, when a major bug, nicknamed Heartbleed, affecting millions of sites was detected in its implementation [2].

An easy way to communicate and understand the dependency of a software project on key developers is the notion of Truck Factor (TF), i.e., the minimal number of developers that the project depends on for its maintenance and evolution [3]. Stated otherwise, if the TF developers abandon the project (e.g., after being hit by a truck) the project maintenance will be heavily affected. Recently, a number of researchers turned their eyes on the importance of studying the TF of software projects, specifically open source ones. Zazworka et al. [4] were the first to propose a heuristic to compute TFs by mining data from version repositories. Cosentino et al. [5] worked on a tool (and novel algorithm) for the same purpose, but targeting git-based repositories. Later, Avelino et al. [6] proposed a heuristic to estimate TFs, based on a code authorship metric. However, the studies going beyond measuring TF towards more profound understanding of what happens when influential TF developers leave the project are still missing. We refer to such a situation as *TF developers detachment* (TFDD).

In this paper, we investigate TFDD with the aim of identifying strategies that help projects to survive. We conduct a mixed-methods study following a sequential explanatory strategy [7]. We start by collecting, curating, and analyzing a dataset of 1,932 popular GitHub projects. Using this dataset, we quantitatively address three research questions: **(RQ1)** How common are TFDDs in open source projects?, **(RQ2)** How often open source projects survive TFDDs? and **(RQ3)** What are the distinguishing characteristics of the surviving projects? These questions will shed light in the prevalence of TFDDs (**RQ1**), project survival (**RQ2**), and evolution of surviving and non-surviving projects (**RQ3**).

Next, we focus on the projects that survive TFDDs and survey 33 developers who assumed the maintenance of a studied project after it was abandoned by its original TF developers. Our qualitative investigation aims to answer three more research questions: **(RQ4)** Do new TF developers perceive risks of project discontinuation?, **(RQ5)** What motivates a developer to assume an open source project after a TFDD situation? and **(RQ6)** What project characteristics most facilitate or hamper the work of recently arrived TF developers? We use this survey to provide qualitative answers about developers' awareness of TFDD occurences (**RQ4**), their motivation to assume the

[1]https://www.instagram.com/about/legal/libraries/
[2]http://opensourcesurvey.org/2017/

978-1-7281-2968-6/19/$31.00 ©2019 IEEE

responsibility for the project (**RQ5**), and enablers and barriers they have experienced while doing so (**RQ6**).

Our contributions are threefold. *First*, we propose a methodology to identify TFDDs by mining software repositories and particularly to identify systems that survive (Section II). *Second*, we show that TFDD is not just a theoretical concept. *Finally*, by surveying TF developers that assumed the maintenance of the surviving systems, we reveal their motivations and difficulties they faced when doing so.

II. Truck Factor

In this section, we first define concepts pertaining to TF. Then, we describe the approach used in the study to calculate TF, identify TFDD and the systems that survived it.

The key definitions used throughout this paper are as follows:

- *Truck factor (TF)* is the minimal number of developers of a project that have to be hit by a truck (or quit) before the project gets in serious trouble [3], [4], [8].
- *TF developers* are the minimal set of developers $\{d_1, d_2, ..., d_n\}$ corresponding to TF. Typically, algorithms estimating TF also compute this set.
- *TF developers detachment* (TFDD) occurs when all TF developers abandon the project.
- *Surviving system* is a system that survives a TFDD, by attracting new TF developers who assume its maintenance.

A. Truck Factor Calculation

To estimate truck factors we use the algorithm proposed by Avelino et al. [6]. The selected TF algorithm initially calculates the degree of authorship (DOA). DOA [9], [10] is a metric reflecting a developer's expertise on each file of the project relatively to the expertise of other developers on the same file. Expertise of a developer on a file is operationalized as the function of whether the developer has created the file, and the number of changes they did on the file compared to changes performed by other developers. Finally, TF estimation relies on the assumption that TF developers are the main authors, i.e., with the highest DOA, of at least 50% of the system's files. We stress that there maybe more than one main author per file, as indicated in the TF algorithm description [6]. The reasons for choosing this algorithm are fourfold: (1) it has the best precision and recall in a recent study comparing three algorithms for estimating truck factors [11]; (2) it scales to large projects with hundreds of contributors; (3) it was validated by surveying the developers of 67 popular GitHub projects [6]; (4) it has a public implementation on GitHub.[3]

B. Identifying Truck Factor Developers Detachments

To search for TFDDs, we first estimate the TF of a system at a time t and verify whether the TF developers abandoned the system before t. We say that a developer *abandoned* a project if their last commit occurred at least one year before the most recent repository commit. Existing studies rely on different thresholds to classify developers inactivity or departure from a project, including three months [12], six months [13], [14], and one year [15], [16]. We experimentally test the sensitivity of five thresholds, in Section III-C, and select the one-year threshold as it is the least sensitive to error.

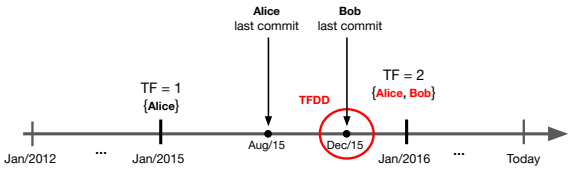

Figure 1. TFDD on *composer/satis*

Example: For the sake of simplicity we do not reproduce the algorithm here, instead we illustrate how it is used in our context. Figure 1 illustrates a fragment of the composer/satis[4] development history[5]. Suppose we first compute the system's TF in January 2015. At this point, the TF estimated by the algorithm equals one, since *Alice* is the (unique) TF developer. As *Alice* is active in January 2015 (she has a commit after this date), no TFDD is observed. When we compute TF in January 2016, TF increases to two, with *Alice* and *Bob* as the TF developers. Moreover, both developers abandoned the project before this date: *Alice* in August 2015 (date of her last commit) and *Bob* in December 2015. Therefore, the developers of composer/satis detached from the project in December 2015.

C. Identifying Surviving Systems

By definition of TF, TFDDs are expected to have a major impact on the evolution of the software project. However, projects can survive such situations. In other words, an occurrence of TFDD does not necessarily imply project termination, e.g., if new developers have taken charge of the project.

We assume a project can be in two states: *Active*, when at least one TF developer is active; and *Inactive*, when all TF developers have abandoned the project. When a TFDD occurs, the system is moved from *Active* to *Inactive*; reversely, the attraction of at least one new TF developer moves the project back to *Active*. Our central object of study are systems with a transition from *Inactive* to *Active* w.r.t. the last occurrence of TFDD; such systems are considered as having *survived* since they became active after their *last* TFDD.

Example: As illustrated in Figure 2, a TFDD occurs in our running example (composer/satis) on December 2015, when both TF developers abandoned the project. Therefore, in this date, the project moved to an *Inactive* state. However, in January 2017, the recomputation of the TF developers resulted in a new developer in this set, *Charlotte*.[6] The attraction of this developer—someone with important contributions to the point of reaching a TF status—moves the project back to the *Active* state. Thus, we say composer/satis survived the TFDD.

[3]https://github.com/aserg-ufmg/truck-factor

[4]https://github.com/composer/satis

[5]To preserve the privacy of the contributors involved, we replace their usernames with pseudonyms.

[6]We compute TFs every year, starting from the repository creation date.

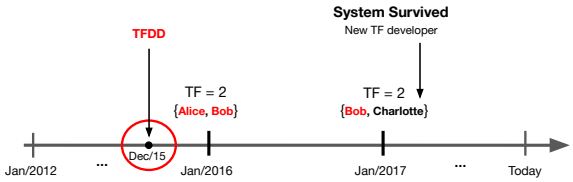

Figure 2. Surviving TFDD on *composer/satis*

Table I
NUMBER OF PROJECTS BY LANGUAGE.

Language	Projects	Language	Projects
Ruby	398 (21%)	PHP	334 (17%)
JavaScript	342 (17%)	Python	297 (15%)
C/C++	335 (17%)	Java	226 (12%)

III. STUDY DESIGN

We adopt a mixed-methods approach and combine a large scale analysis of version control repository data with a survey. Mixed-methods are appropriate for the pragmatic stance common in software engineering research [7].

A. Dataset & Preprocessing

To perform the quantitative part of the study, we build a dataset with GitHub projects. Initially, we focus on six programming languages with the largest number of GitHub repositories: JavaScript, Python, Ruby, C/C++, Java, and PHP. We select the top-500 most starred repositories (excluding forks to avoid including the same project multiple times) for each of those languages at the moment of analysis. We focus on popular projects to ensure the quality of the data, so that the collected projects are relevant to the OSS community, and to avoid including personal projects in our dataset [17], [18].

To safeguard the quality of the dataset we filter the resulting collection of 3,000 GitHub repositories. We explicitly address well-known "perils of mining GitHub" [17]. We exclude (a) projects that did not use GitHub exclusively during their entire history and lost part of their development history when migrated to GitHub, (b) projects that do not have sufficient historical data for the TF computation, and (c) projects that are not software units or are explicitly labeled as unmaintained. To identify projects with evidence of loss of part of their development history we filter out repositories where more than 50% of the files are added in less than 20 commits in the beginning of their development. By applying this filter, we exclude 677 projects. As our approach to identify TFDD requires at least two years of historical data, we filter out 338 projects with less than two years of development activity. To apply the last filter we manually inspect the project descriptions and exclude 53 projects. Among others, we found repositories containing books, awesome-lists (i.e., sets of suggested books, links, etc.), and technology code samples. The resulting dataset is composed of 1,932 ($= 3,000 - 677 - 338 - 53$) projects.

As shown in Table I, most projects are implemented in Ruby (398 projects, 21%). On the other side, Java is the language with fewest projects (226 projects, 12%). Figure 3 shows violin plots with the distribution of the number of developers, source code files, commits and stars per project (please note the logarithmic scale). The median values are indicated inside the violin plots. We conclude that the dataset constructed typically includes large systems, both in size and in number of developers, and that the systems also are popular (number of stars) and have a large number of commits.

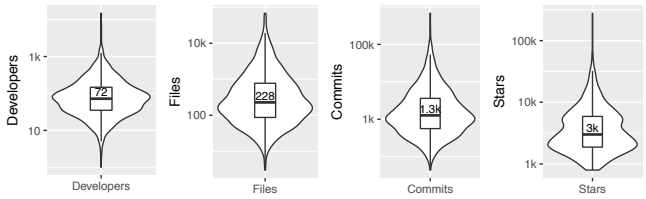

Figure 3. Distribution of the number of developers, commits, files, and stars.

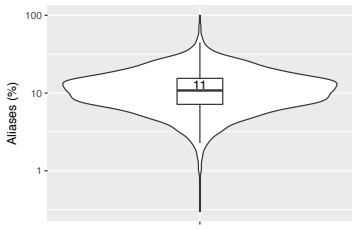

Figure 4. Percentage of aliases in each project

B. Aliases Resolution

The correctness of TF computations highly depends on the set of distinct developers. However, developers do not necessarily use only one alias (name or e-mail address) when contributing to a project [19], [20], [21]. Therefore, it is important to detect and resolve aliases among the developers of the 1,932 projects in our dataset. Rather than using heuristics advocated in previous works to detect aliases [19], [20], [21], we use a feature of the GitHub API that maps an e-mail address in the commit header to a GitHub user. Using this feature, we mapped each developer of each system to their GitHub account; d_1 and d_2 are considered aliases when they are mapped to the same GitHub account. As a downside, this approach does not handle the cases where developers have multiple GitHub accounts. Figure 4 shows a violin plot with the percentage of aliases in each project. The median percentage of aliases in a project in our dataset is 11%.

C. Abandoner Threshold Sensitivity Analysis

The selected threshold value to identify developers *abandoning* projects can impact our results. We therefore test the sensitivity to error of five different threshold values, i.e., 3 months, 0.5 year, 1 year, 1.5 year and 2 years, to select the most appropriate threshold. We first gather each TF developer's commit activity and then measure the elapsed time delta between consecutive commits. For each TF developer with N commits, we compute $N-1$ inter-commit time deltas.

Table II
THRESHOLD SENSITIVITY

T_i	$P(T_i)$	$impr(T_i, T_{i-1})$	harmonic_mean
3 months	0.38	-	-
6 months	0.59	0.35	0.44
1 year	0.82	0.55	**0.66**
1.5 year	0.91	0.50	0.64
2 years	0.95	0.46	0.62

Since each time delta represents the time elapsed between commits, developers should never be classified as abandoners by a threshold. In other words, appropriate thresholds should optimally have zero error, meaning that they will never erroneously classify a developer as an abandoner, as by definition she has at least one subsequent commit.

To assess the error sensitivity of a list of thresholds TS where $TS = \langle T_1, T_2, ..., T_N \rangle$ such that $T_i < T_{i+1}$ for $i \in 1...N-1$, we used the precision and improvement metrics, as well as their harmonic mean. Precision $P(T_i)$ of a threshold T_i is defined as the percentage of developers that T_i has zero error, i.e., T_i never classifies them as abandoners. Improvement $impr(T_i, T_{i-1})$ of T_i over the smaller threshold T_{i-1} is defined as the number of developers that T_i has zero error, while T_{i-1} erroneously classifies as abandoners over the total number of developers that T_{i-1} erroneously classifies as abandoners. In practice, $impr(T_i, T_{i-1})$ measures how many errors of T_{i-1} were corrected by T_i. The harmonic mean between precision and improvement is defined as $\frac{2*P*impr}{P+impr}$.

Table II presents the sensitivity analysis results for the five threshold values considered. The precision results indicate that a certain amount of error is introduced regardless of the threshold, e.g., even a 2-year threshold produced an error of 5%. On the contrary, the largest improvement is achieved by the 1-year threshold (55%) over the 6-months threshold, indicating that more than half of the errors made by the 6-month threshold were fixed by the 1-year threshold. Overall, the 1-year threshold achieves the highest harmonic mean value (66%) compared to the other thresholds. We therefore use the 1-year threshold in our experiments to determine if a developer has abandoned the project after their last commit.

IV. SEARCHING FOR TFDDS AND SURVIVING PROJECTS

Prior to analyzing TFDDs, we estimate the TFs for 1,932 projects in our dataset using the algorithm of Avelino et al. [6]. We clone the project repositories and hereby provide statistics based on the most recent snapshot of the considered repositories; the TF analysis is performed yearly since the first commit of each project to answer the first three research questions. Figure 5 presents a histogram with the TF results. As we can observe, most projects have a low TF: e.g., for 57% projects TF equals 1, while less than 6% have a TF higher than 5. The highest TF is 26, computed for edx/edx-platform, which is the software platform that supports edX massive open online courses. Our findings concur with the earlier results of Avelino et al. [6] that reported that 65% of the evaluated systems have TF $\leq$ 2, based on a sample of 133 popular GitHub projects.

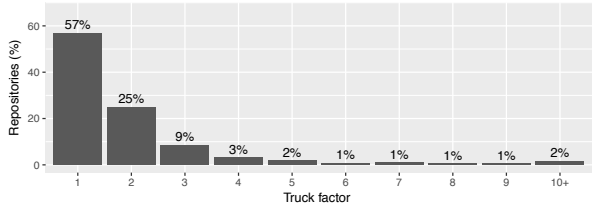

Figure 5. TF of the 1,932 projects in our dataset

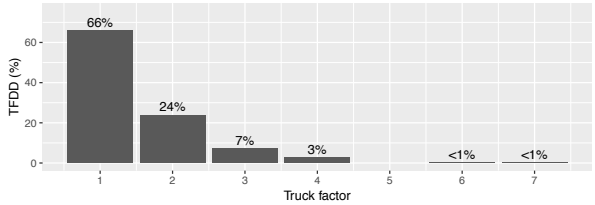

Figure 6. Projects facing TFDDs

> Most open source projects have low TFs. In a sample of 1,932 projects, 57% have TF = 1 and 25% have TF = 2. The highest TF in our sample is 26 developers.

In the remainder of this section, we describe a quantitative exploration of the collected data, aiming to answer *(RQ1)*–*(RQ3)*. We start by assessing whether TFDDs indeed happen in open source development *(RQ1)*. Assuming that TFDDs indeed occur, *RQ2* takes a step further and investigates how often projects overcome such situations. Finally, assuming we find projects that survived their TFDDs, we compare them with other projects that did not have the same fate *(RQ3)*. The goal is to identify characteristics that might help projects to overcome the loss of TF developers.

RQ1) How common are TFDDs in GitHub projects?

We identify TFDDs in 315 projects, 16% of our dataset. Most of the projects faced only one TFDD situation (88%). However, some projects faced two (11%) or even three ($<$ 0.1%) TFDDs. Figure 6 shows the percentage of TFDDs grouped by TF. As expected, most TFDDs are observed in systems with a small TF, e.g., 66% of TFDDs happens in projects with a TF equal to one. This means that most projects that are in a TFDD situation are maintained by one core developer; it remains to be seen if most projects are in such a situation only once because they become obsolete or because they survive it and never face one again. We further investigate project survival after TFDDs in Section IV.

In contrast, projects found in a TFDD situation only twice have a TF higher than four: etsy/logster ($TF = 7$) and PointCloudLibrary/pcl ($TF = 6$). etsy/logster is a small project, with 13 files and 117 commits when the TFDD was observed. By contrast, PointCloudLibrary/pcl is a large project, with 9,568

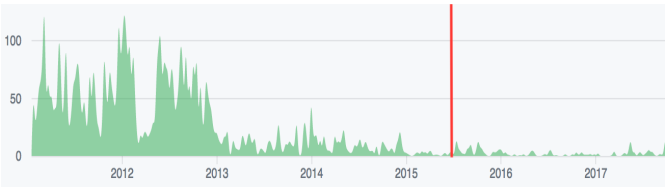

Figure 7. Contributions to PointCloudLibrary/pcl over time (screenshot from GitHub). A TFDD occurred at June, 2015 (vertical red line).

commits and 2,204 files at TFDD time. All TF developers started contributing to this project in the first year of its development (2011), but abandoned the project before 2015. To show the impact of their departure, Figure 7 shows a screenshot with the contributions to PointCloudLibrary/pcl, as available on its GitHub page[7]. Most contributions happened before June, 2015, when the project faced a TFDD (vertical red line, in the figure). This was the date of the last commit of one of the TF developers. The commits of the other five TF developers all happened before May, 2014. Although PointCloudLibrary/pcl has had financial support from a non-profit organization,[8] as indicated in the project's README page, the site and social network accounts of this organization do not receive updates since 2014, which is close to the TFDD date.

> Truck Factor developers detachment is not merely a theoretical concept: 16% of the projects faced at least one TFDD; 66% of these TFDDs happened in systems with TF=1, which are 55% of the projects.

Figure 8 shows the age of the repositories with TFDDs, considering their creation date on GitHub. As we can see, most projects (71%) have between 4 and 7 years of development. Figure 9 shows when these TFDDs happen, in terms of number of development years and counting only the first TFDD, for projects with multiple TFDDs. As we can observe, there is a concentration of TFDDs in the first years of development; 59% took place in the first two years of development. In fact, in some cases the TF developers abandoned the projects some time after the repository creation, e.g., in 23 projects the TF developers abandoned the projects in the first six months.

> 59% of the TFDDs happened in the first two years of development; but 71% of the projects with TFDDs have now between 4 and 7 years of development.

RQ2) How often open source projects survive a TFDD?

A project survives if it survives the last observed TFDD. In total, 128 projects (out of 315 projects) overcome their TFDDs, which represents a survival rate of 41%. In most cases (86%) we detected that only one new TF developer was attracted to the project and was responsible for its survival.

[7]https://github.com/PointCloudLibrary/pcl/graphs/contributors
[8]http://www.openperception.org

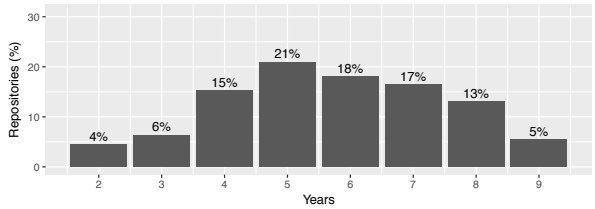

Figure 8. Age of the repositories with TFDDs

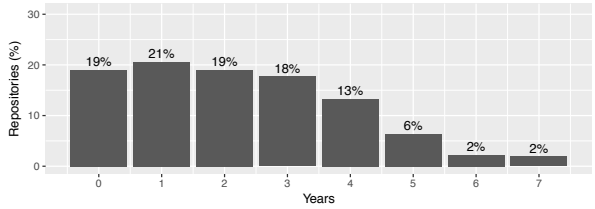

Figure 9. When do TFDDs happen (counting from the repositories creation)

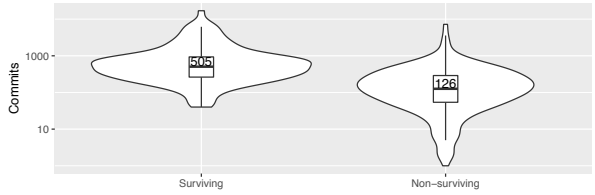

Figure 10. Number of commits after the last observed TFDDs

However, there are cases where two (12%) or even three (2%) new TF developers were attracted to the projects. Additionally, in 64% of these cases the attraction occurred in the first year after the TFDD, while 23% occur in the second year, 10% in the third year and 2% in the fourth year. As expected, it becomes more difficult to attract new TF developers to assume project maintenance throughout the years.

> It is possible to recover from TFDDs: 41% of the projects survived their last observed TFDD, usually by attracting a single new TF developer (86%).

A developer is called a *newcomer* if their first commit occurs after the last observed TFDD. Otherwise, they are an *old-contributor*. In most surviving projects (52%), the new TF developers are all *old-contributors*. However, a significant part of the projects survived with the help of *newcomers* (41%) or by attracting both newcomers and old contributors (7%).

> Newcomers are crucial to recover from TFDDs. They contributed to recovery of 48% of the surviving projects.

RQ3) How surviving projects differ from non-surviving ones?

Figures 10 and 11 show respectively the distribution of the absolute number and the percentage of commits after the last detected TFDD in each surviving project (128 projects)

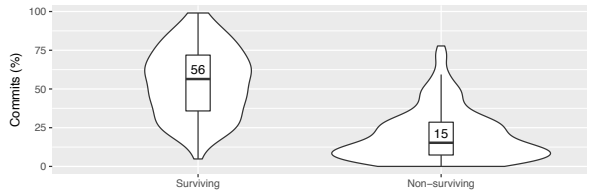

Figure 11. Percentage of commits after the last observed TFDDs

and also in the non-surviving ones (187 projects). Before discussing these figures, we stress that TFDD should have a major impact on project maintenance and evolution, but this does not necessarily mean that the project maintenance has ceased afterwards. Therefore, it is possible to observe commits after TFDDs even in non-surviving systems. However, these commits are *performed by minor contributors* and do not affect the TF developer set. This means that the projects continue to be at risk even in the presence of commits after a TFDD.

The violin plots in Figures 10 and 11 show a clear difference between surviving and non-surviving systems. The surviving systems have 505 commits (56%) after the last detected TFDD, whereas the non-surviving ones have only 126 commits (15%), considering the median values. The third quartile measures are 949 commits (72%) and 289 commits (29%) for surviving and non-surviving projects respectively. These differences are confirmed using the one-sided version of the Mann-Whitney test ($p = 5.02 \times 10^{-22}$ and $p = 2.04 \times 10^{-32}$ for the number and percentage of commits after the last TFDD respectively). The effect size, measured by Cliff's delta [22] and using the intervals of Romano et al. [23], is *large* in both cases: $d = 0.64$ for the number of commits, and $d = 0.79$ for the percentage of commits after the last TFDDs.

We also explore the differences, if any, between surviving and non-surviving systems w.r.t different factors in order to reveal if such factors can provide insights related to project survival. Figure 12 shows violin plots with the distributions of the number of developers, commits and files, and project age measured in days of the surviving and non-surviving projects. All values refer to the date of the studied TFDDs. We test the differences between surviving and non-surviving projects using two-sided Mann-Whitney tests and by visually confirming the differences using the visualizations of Figure 12. Since we consider different aspects of the same projects we adjust the p-values to control for multiple comparisons using the method of Benjamini and Hochberg [24]. We select this method as it is more powerful than the alternative techniques.

Interestingly, the surviving projects have less developers than the non-surviving ones (32 vs 47, median values, $p = 2.2 \times 10^{-4}$). They also have less commits (384 vs 694, median values, $p = 2.6 \times 10^{-4}$) and less files (54 vs 85, $p = 4.7 \times 10^{-2}$). However, the effect size is *negligible* for number of files ($d = 0.13$) and *small* for number of commits ($d = 0.25$) and developers ($d = 0.26$). Surviving projects are also younger at the time of the TFDD (1095 vs 1460 days, median values, $p = 3.4 \times 10^{-7}$) with a *medium* effect

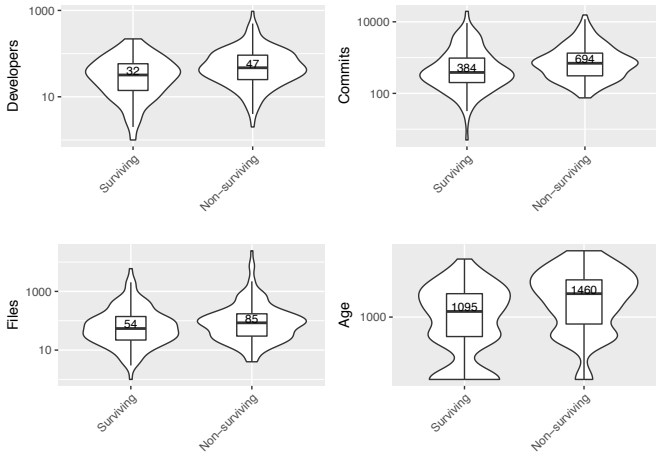

Figure 12. Number of developers, commits and files, and project age for surviving and non-surviving projects (at the date of the studied TFDDs)

size ($d = 0.37$). We conjecture that non-surviving projects are either feature-complete (as they are more mature) or that they failed to attract new developers to assume their maintenance. However, it is important to consider that even feature-complete systems require corrective maintenance for fixing bugs [25]. It is thus uncommon that a project is both feature-complete and bug-free thus not requiring any further maintenance.

> At the moment of the TFDDs, we found no major difference between surviving and non-surviving projects, in terms of number of developers, commits, and files. On the contrary, we found that surviving projects are younger at TFDD time compared to the non-surviving ones.

V. Survey with new TF Developers

In this section, we report the results of a survey with the new TF developers of the surviving projects, i.e., developers that played a major role in the maintenance of these systems after the identified TFDDs, to answer to the last three research questions. The intention of ***RQ4*** is to check whether the developers perceived the projects being at risk, before making the contributions that led them to reach a TF developer status. ***RQ5*** investigates how OSS managers should proceed to attract new TF developers to their systems. Finally, the intention of ***RQ6*** is to shed light on programming and management practices that should be promoted (or avoided) in OSS development.

A. Survey Design

For each surviving project (128 projects), we select their *new* TF developers. After excluding the ones without a public and valid e-mail address, we have identified 144 potential participants. We sent an e-mail to these developers with four questions: (1) Did you think that [project] was at risk of being discontinued before deciding to make major contributions to its continuation? (2) Why did you decide to make these contributions? (3) What project characteristics and practices

helped you to make these contributions? (4) What were the main barriers you faced when making these contributions?

From the 144 e-mails we sent, four returned due to an invalid address. In total, we received 33 answers, representing a response rate of 24%; this rate is higher than what has been achieved by previous papers [26], [27]. To process the answers, we rely on Thematic Analysis [28], which consists of (i) initial reading of the developer responses; (ii) generating initial codes for each response; (iii) searching for themes; (iv) reviewing the themes to find opportunities for merging among themes; and (v) defining and naming the final themes. The three first steps were performed independently by two authors of this paper; after that, in a meeting, they performed the two last steps, by checking and revising the final themes.

RQ4) Do new TF developers perceive risks of project discontinuation?

We started the survey with the question about the perception of the discontinuation risks, according to the new TF developers. The purpose of this question is twofold. First of all, it serves as an additional validation of both our approach concerning the identification of TFDDs, i.e., whether or not TFDDs capture the project being at risk of discontinuation at the time, as well as the importance of the TFDDs: if developers believe that the project is at risk of discontinuation after a TFDD, then further evolution of the project is indeed threatened. Second, this question assesses awareness of the new TF developers; awareness of the context and therefore of their tasks helps team coordination in software development [29].

Table III summarizes the results for this question. Most respondents (18 developers, 60%) agreed that the projects were facing risks of discontinuation. Examples of positive answers include: *"when I thought the project would die, I started making contributions to it once again"* (D8); and *"yes, otherwise the project would been completely abandoned"* (D9). Furthermore, we classify as a partial agreement five cases (17%) where the developers reported problems in the projects, but were not clear about their severity, or mentioned the problem was mitigated by another developer who also entered in the TF set. As examples, we have these answers: *"[the] development had slowed"* (D14); and *"A new primary developer stepped in and took responsibility of the project after the original developer left."* (D32). Indeed, the developer mentioned by D32 was also identified as a new TF developer in our study; therefore, we contacted him for our survey, but unfortunately did not receive a reply.

Finally, six developers (20%) answered that they did not perceive the projects as being at risk. Usually, these developers were succinct in their answers (just answering *no*, for example). Remarkably, among the negative answers, one developer mentioned the project is supported by a major software company, which contributes to reduce the discontinuation risks, in his opinion: *"this open source project is actually backed by a for profit company, so the project didn't risk being abandoned"* (D24). Finally, we classify one answer as unclear (3%), because it is not related to the provided question. Four respondents did not answer this survey question.

Table III
DID YOU PERCEIVE THE PROJECTS AT RISK?

Yes	Partially	No	Unclear
18 (60%)	5 (17%)	6 (20%)	1 (3%)

Table IV
MOTIVATIONS TO CONTRIBUTE

Motivations	Devs	%
Because I was using the project	17	53
To contribute to an open source project	11	34
To avoid the project discontinuation	5	16
I have interest on the project area	4	13
I get paid to contribute	4	13
To improve my own skills	3	9
I have the skills required by this project	3	9
It is a successful project	3	9
Others	7	22

> 77% of the new TF developers were (partially) aware of the risks faced by the surviving systems, before making the contributions responsible for the project recovery.

RQ5) What motivates a developer to assume an open source project after a TFDD situation?

With this question, we aim to reveal the reasons that motivated the new TF developers to make their major contributions to the projects. Table IV summarizes the main reasons mentioned by the surveyed developers. **Because I was using the project to address my personal or professional needs** is the most common reason mentioned by 17 participants (53%). Answers referring to this reason include *"I used the [project] in my own products and was struggling with a few bugs so I decided to fix them and contribute back"* (D14); *"mostly because I used [project] heavily and was asked for documentation and improvements from within my company"* (D16); and *"I used the project professionally and was in a position to provide some level of support as part of my job"* (D23). **To contribute to an open source project** is the second most common reason mentioned by 11 participants (34%); **to avoid the project discontinuation** is mentioned by five developers (16%), e.g., *"I was the only additional contributor on the project so I was the only person capable of keeping it alive"* (D7); One respondent did not answer this question.

> The developers responsible to reactivate the maintenance of the surviving projects were motivated by their own usage of the projects (17 developers, 53%). They also intended to contribute back to an open source community (34%) or avoid the project discontinuation (16%).

Table V
CHARACTERISTICS THAT HELPED NEW TF DEVELOPERS

Type	Characteristics	Devs	%
Human/ Social	Friendly and active owners/members	12	41
	I liked/knew the project	3	10
Technical	Programming language	4	14
	Well-known SE principles	4	14
	Pull based development	4	14
	Continuous integration	2	7
	Clean and well-designed code	2	7
	Code revision	1	3
Others	Main repository access	3	10
	Job support	2	7
	Small or simple project	2	7
	Open source license	1	3

Table VI
BARRIERS FACED BY NEW TF DEVELOPERS

Type	Barriers	Devs	%
Human/ Social	Lack of time	7	26
	Lack of experience	3	11
	Unfriendly maintainers	2	7
Technical	Need to keep backward compatibility	4	15
	Lack of well-known SE principles	1	4
Others	Lack of access to the main repository	5	19
	Large number of pending issues	3	11
	No financial support	2	7
No barriers	-	4	15

RQ6) What project characteristics most facilitate or hamper the work of recently arrived TF developers?

We start with the project characteristics that facilitated the attraction of the new TF developers. As presented in Table V, we organize these characteristics in three groups: human and social characteristics (15 answers), technical characteristics (17 answers), and other characteristics (8 answers). The most mentioned human and social characteristic is the presence of friendly and active project owners or members (12 answers). As examples, we have these answers: *"it has been [dev-name]'s kindness to my first contributions and his help to me, and later other cool developers' support"* (D6); *"the responsiveness of the existing maintainer was the key factor to my ongoing contributions"* (D11). Among others, technical characteristics include the usage of a specific programming language (4 answers) or following well-known software engineering principles and practices (4 answers). The last category groups factors like permission to access the main repository (3 answers) and financial support by a company (2 answers). Four respondents did not answer this question.

> The characteristics that helped on the attraction of new TF developers have a social, technical or external nature. Friendly and active maintainers is the most mentioned facilitator, indicated by 12 developers (41%).

To complement the answer to **RQ6**, we also asked the new TF developers about the barriers they faced when making the contributions that led them to achieve a status of TF developer. As in the case of the first part of the question, we organize the answers mentioned by the participants in three groups: human and social barriers (12 answers), technical barriers (5 answers), and other barriers (10 answers). Table VI presents the answers in each group. As we can observe in this table, most answers denote human and social barriers. Particularly, *lack of time* is the most common barrier mentioned by the survey participants (7 answers). As examples, we have these answers: *"I have other projects to maintain."* (D3); and *"time is always an issue, especially because the range of features is fairly wide"* (D23). Technical barriers include the requirement to *keep backward compatibility and do not introduce bugs* (4 answers) and the lack of solid software engineering principles (1 answer). Another barrier commonly mentioned by the participants is the *difficulty to obtain access to the main repository* (5 answers). The participants justify the need to obtain this access because the *maintainers are absent* (D2, D12) or *the project was abandoned* (D8, D9). Four developers mentioned they faced no barriers at all. Six developers did not answer this question.

> Human and social barriers are the most common ones faced by new TF developers; particularly, lack of time is the most common barrier.

VI. DISCUSSION

Truck factor is not only a theoretical metaphor: In OSS development, it is possible to argue that TF is just a theoretical scenario, since the code is public and others can assume the maintenance work if the key developers abandon the project. In fact, one of the participants of the survey provides an argumentation in this direction: *"it's open source, if people want to use it, they will use it. If it's missing features they really want/need, they will submit PR's, or fork and maintain their own copy."* (D30). Undoubtedly, if the code is public on GitHub, anyone has the legal permission (according to the project's license) to collaborate with or fork the project. Moreover, GitHub provides many useful instruments to facilitate this process, like easy forking or pull requests. Despite that, our study shows that even popular projects may fail to attract new contributors after being abandoned. More precisely, only 41% of the projects have fully recovered the maintenance activity after the TFDDs studied in our work. We hypothesize that assuming the maintenance of an open source project is a complex task, which requires time, technical and social skills and familiarity with the project domain. Many projects therefore do not succeed to find developers with this profile and face serious maintenance problems or even fail after being abandoned by their TF developers. We stress that projects cannot rely only on peripheral contributions to survive as such contributions are complementary to the ones of TF developers and cannot fully support the project maintenance [30], [31].

Interestingly, we also found arguments in the opposite direction, stating that TF is a less important concern in software projects backed by a company regardless of the project being

open source. We received at least two answers hinting in this direction, as this one: *"Most of the questions are not relevant because [project] is actually a large project with formal sponsorship by [company]"* (D33). In other words, these developers consider that TF is a real concern only in projects without financial support, as is the case of most open source projects.

Regardless of the point of view, our work showed the implications of TF developer abandonment in project evolution. It is therefore essential, for project key maintainers either to strive to increase the number of TF developers or to provide an alternative backing, e.g., company-based support, in order to prevent or reduce the chances of TFDDs.

How to overcome TFDDs: Although we show that TFDD situations are a reality in OSS development, we also found that it is possible to survive and recover the maintenance after attracting new developers to the TF set. By surveying these new TF developers, we shed light on two key characteristics of the surviving projects.

First, the surveyed developers decided to assume the maintenance of these projects motivated by their own needs, since they were using the projects and require new features or fix existing bugs. This finding suggests a connection between the number of users of an OSS project and its resilience to TFDDs. Particularly, 53% of the TF developers surveyed in our study were attracted because they were earlier users of the projects and therefore had personal interests in avoiding their failure.

Second, human and social factors have a key role on attracting new TF developers. According to the survey participants, 51% of the factors that helped in their attraction are social in nature; and 44% of the barriers faced in this process are also human and social ones. The importance of human and social barriers to technical contributions was also observed by Palomba et al. [32]. Our findings, therefore, confirm the importance of human and social factors in OSS development. This is particularly the case if most contributions are voluntary, as indicated by one of the survey respondents: *"There is no authority over the top that chooses who will work on what. We are all contributing during our "free" time, for only the "enjoyment" of it. So, we sort of contribute only where it "feels" good"* (D26).

Our work sheds light on the origin of new TF developers that assume project maintenance. In turn, this information can be utilized to prevent or limit the prevalence of TFDDs. TF developers that consider leaving a project, can identify individuals that can serve as potential new TF developers and prevent the discontinuation of the project. Our study shows that such individuals can be either volunteers using the project or company-based developers that have had prior contributions to the project, in the case that a company uses the project.

VII. Threats to validity

Construct validity: Firstly, our results depend on the accuracy of TF computations. To mitigate this threat we used the TF algorithm that presents the best accuracy, as pointed by a recent comparative study [11]. Nonetheless, we quantified the risk stemming from the choice of the TF algorithm by measuring the distribution of the percentage of files of the new TF developers for the 128 surviving systems. If the new TF developers work on few files, then they should not be considered TF developers suggesting imprecision of the algorithm. We observed that new TF developers have worked on a substantial share of a system's files ($Q_2 = 41\%$) increasing our confidence in the algorithm.

Identification of TFDDs depends on the selected abandonment threshold. As there is no consensus on this threshold in the literature, we experimentally test five different thresholds (Section III-C) to select an appropriate threshold. However, using a different threshold might lead to different results. The TF measures may also vary due to possible developer aliases. We mitigate such a threat by carefully handling common sources of aliasing (Section III-B).

Another threat concerns our definition of TFDD as occurring if *all* TF developers abandon the project. To evaluate the accuracy of this definition, we rely on the survey. While we are aware of the limitations of this evaluation strategy, we observe that more than 75% of the respondents confirmed their (partial) awareness that the project was indeed at risk. Therefore, we conclude that the effect of this threat is low.

Finally, we see a system as an evolving artifact, requiring continuous change. For mature software that does not require changes, a TFDD analysis can thus lead to erroneous results. However, this is uncommon as based on Lehman's software evolution laws [33], software must be continuously adapted or it becomes progressively less satisfactory. To quantify this threat we manually inspected the README files of the 187 non-surviving systems. Although we found one system (defunkt/jquery-pjax) described as feature-complete, it is still maintained when bugs need to be fixed.

Internal Validity: Identification of TFDDs uses data from the entire development history of a project, and is, hence, sensitive to the partial loss of history, e.g., due to corrupted migration from another version control system. We mitigate this threat by excluding projects with evidence of a corrupted migration to GitHub (Section III-A). We also exclude non-software projects, such as books and tutorials.

External Validity: Our dataset was carefully selected from popular projects on GitHub. However, our findings cannot be generalized to other projects and particularly to closed-source projects. Indeed, our survey results suggest that TFDDs in the context of software with financial support might have very different characteristics.

VIII. Related work

Truck factor is a concept defined by the agile community to assess knowledge concentration in software projects. As the concept initially lacked a formal definition, the first works in this area focused on proposing algorithms to compute truck factors. The first algorithm to this purpose was proposed by Zazworka et al. [4]. After that, it was used by Ricca et al. [34] and Torchiano et al. [35], respectively, to investigate the presence of "heroes" in open source projects

and to investigate threshold values to use when computing truck factors. However, Zazworka's algorithm suffers from scalability problems [36], [37], which limits its applicability to real systems. To address these problems, new algorithms were proposed by Cosentino et al. [5], Rigby et al. [38] and Avelino et al. [6]. Ferreira et al. [11] compared these three algorithms and concluded that the latter algorithm is the most accurate one. None of the aforementioned works investigated whether TFDDs really occur and what happens with open source projects afterwards.

Truck Factor can be considered as a particular case of turnover, involving the principal developers of a project. Turnover of developers in general is a well-studied phenomenon in software engineering. Foucault et al. [14] report the negative impacts of turnover in the internal quality of five open source projects. Rigby et al. [38] and later Nassif et al. [39], profiled the knowledge loss induced by developers who leave a software project and provide tools to help large projects to assess the risk of turnover. Hilton and Begel [40] recently studied internal turnover in a major software company, with more than 30K employees. By surveying a sample of 374 of such employees, they reveal what causes engineers to consider leaving their teams, why they leave, how they learn about new teams, and how they decide which team to join. Lin et al. [13] conducted a similar study, but with focus on five open source projects. They show that developers are retained when they (i) start contributing to the projects earlier, (ii) maintain both code developed by others and their own code, and (iii) mainly code instead of writing documentation. Vasilescu et al. have shown that presence of developers with very different durations of engagement on GitHub increases the likelihood of turnover [41]. Qiu et al. have observed that while women tend to disengage from open source projects faster than men, attachment of women to open teams with regard to diversity of information increases their chance of prolonged engagement more than the chances of men [42]. Recently studies of contributor disengagement have been conducted by Miller et al. [43] and Iaffaldano et al. [44].

Motivations and barriers to contribute to open source systems were previously investigated for different developer profiles: developers that have only one patch accepted [45], developers with few contributions and no intention to become an active project member [46], newcomers [47], [48], and core developers [49]. Regarding the reasons that motivate developers to contribute to open source, some of these studies also show that core developers are motivated by their personal needs, as we concluded for the specific case of new TF developers. By contrast, one-time contributors and casual contributors are mainly motivated by the need to fix minor bugs. Lack of time is a common barrier to contribute, mentioned by core developers, one-time contributors, and casual contributors. It was also commented by the new TF developers surveyed in our study. Steinmacher et al. [47] defined a conceptual model composed of 58 barriers that may hamper newcomers' first contributions. They list and classify these barriers, but do not provide insights on their frequency. Coelho et al. [50], report a survey with the maintainers of 104 failed open source projects, i.e., projects that are not maintained anymore. According to their survey, the most common reasons for open source project failures are the appearance of a strong competitor, obsolescence, and lack of time or interest of the project owners. By comparing the factors that attract new contributors with our results about attracting new TF developers, we found that new TF developers are motivated by their own use and need to save the projects (53%) in contrast to new, casual or one-time contributors. On the other hand, our results about the importance of human and social barriers in OSS comply with related work, e.g., reception issues [47] and friendly and active owners/maintainers (12 out of 29 respondents in our survey). Identification of welcoming projects with friendly maintainers is related to such topics as community health [51], presence of codes of conduct [52] and emotions expressed in developer communication [53].

IX. Conclusion

In this paper, we presented an in-depth investigation of the occurrence of TFDDs in open source projects, i.e., the abandonment of a project by its principal developers. We showed that TFDDs are not only a metaphor, but they indeed happen in open source projects (in 16% of such projects, at least in our sample of 1,932 GitHub projects). Additionally, we showed that projects survive such situations, by attracting new core contributors (41% of the projects survived a TFDD, in our sample). Finally, we reveal the motivations that led these developers to take over the studied projects, after the projects faced a TFDD. We also reveal the principal enablers and barriers faced by these developers during this process. This list of enablers and barriers are especially useful and should be considered to build development communities that are more attractive to new contributors.

As future work, we envision the design, implementation and evaluation of tools to assess the risks faced by an open source project, in case it is abandoned by its TF developers. This assessment is particularly important to the users of such projects. We also see space to investigate recommenders of TF developers for a system, based for example on their own usage of the projects. Finally, open source communities should be made aware of successful cases of projects overcoming TFDDs, as we report in our paper, and motivate developers to actively contribute to projects at risk.

As a final note, we provide a replication package with the results of our analysis as well as the survey's answers, repositories data, and scripts used in the paper. The replication package is available at https://doi.org/10.5281/zenodo.2546008.

Acknowledgments

We thank all respondents of our survey. This work was partially supported by the Excellence of Science Project SECO-Assist (O015718F, FWO - Vlaanderen and F.R.S.-FNRS), the FRQ-FNRS collaborative research project R.60.04.18.F SECO-Health, CAPES (131987/2016-01) and CNPQ (140205/2017-9).

REFERENCES

[1] N. Eghbal, "Roads and bridges: The unseen labor behind our digital infrastructure," Ford Foundation, Tech. Rep., 2016.

[2] Z. Durumeric, F. Li, J. Kasten, J. Amann, J. Beekman, M. Payer, N. Weaver, D. Adrian, V. Paxson, M. Bailey, and J. A. Halderman, "The matter of heartbleed," in *IMC*, 2014, pp. 475–488.

[3] L. Williams and R. Kessler, *Pair Programming Illuminated*. Addison Wesley, 2003.

[4] N. Zazworka, K. Stapel, E. Knauss, F. Shull, V. R. Basili, and K. Schneider, "Are developers complying with the process," in *ESEM*, 2010.

[5] V. Cosentino, J. L. C. Izquierdo, and J. Cabot, "Assessing the bus factor of Git repositories," in *SANER*, 2015, pp. 499–503.

[6] G. Avelino, L. Passos, A. C. Hora, and M. T. Valente, "A novel approach for estimating truck factors," in *ICPC*, 2016, pp. 1–10.

[7] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, "Selecting Empirical Methods for Software Engineering Research," in *Guide to Advanced Empirical Software Engineering*, F. Shull, J. Singer, and D. I. K. Sjøberg, Eds. Springer, 2008, pp. 285–311.

[8] M. Lavallee and P. N. Robillard, "Why Good Developers Write Bad Code: An Observational Case Study of the Impacts of Organizational Factors on Software Quality," in *ICSE*, 2015, pp. 677–687.

[9] T. Fritz, J. Ou, G. C. Murphy, and E. Murphy-Hill, "A degree-of-knowledge model to capture source code familiarity," in *ICSE*, 2010, pp. 385–394.

[10] T. Fritz, G. C. Murphy, E. Murphy-Hill, J. Ou, and E. Hill, "Degree-of-knowledge: modeling a developer's knowledge of code," *Transactions on Software Engineering and Methodology*, vol. 23, no. 2, pp. 14:1–14:42, 2014.

[11] M. Ferreira, M. T. Valente, and K. Ferreira, "A comparison of three algorithms for computing truck factors," in *ICPC*, 2017, pp. 207–217.

[12] E. Constantinou and T. Mens, "Socio-technical evolution of the Ruby ecosystem in GitHub," in *SANER*, 2017, pp. 34–44.

[13] B. Lin, G. Robles, and A. Serebrenik, "Developer turnover in global, industrial open source projects: Insights from applying survival analysis," in *ICGSE*, 2017, pp. 66–75.

[14] M. Foucault, M. Palyart, X. Blanc, G. C. Murphy, and J.-R. Falleri, "Impact of developer turnover on quality in open-source software," in *FSE*, 2015, pp. 829–841.

[15] D. Izquierdo-Cortazar, G. Robles, F. Ortega, and J. M. Gonzalez-Barahona, "Using software archaeology to measure knowledge loss in software projects due to developer turnover," in *HICSS*, 2009, pp. 1–10.

[16] E. Constantinou and T. Mens, "An empirical comparison of developer retention in the rubygems and npm software ecosystems," *Innovations in Systems and Software Engineering*, vol. 13, no. 2-3, pp. 101–115, 2017.

[17] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "An in-depth study of the promises and perils of mining GitHub," *Empirical Software Engineering*, 2015.

[18] N. Munaiah, S. Kroh, C. Cabrey, and M. Nagappan, "Curating github for engineered software projects," *Empirical Software Engineering*, vol. 22, no. 6, pp. 3219–3253, 2017.

[19] E. Kouters, B. Vasilescu, A. Serebrenik, and M. G. J. van den Brand, "Who's who in Gnome: Using LSA to merge software repository identities," in *ICSM*, 2012, pp. 592–595.

[20] M. Goeminne and T. Mens, "A comparison of identity merge algorithms for software repositories," *Science of Computer Programming*, vol. 78, no. 8, pp. 971–986, 2013.

[21] I. S. Wiese, J. T. da Silva, I. Steinmacher, C. Treude, and M. A. Gerosa, "Who is who in the mailing list? comparing six disambiguation heuristics to identify multiple addresses of a participant," in *ICSME*, 2016, pp. 345–355.

[22] R. J. Grissom and J. J. Kim, *Effect Sizes for Research: A Broad Practical Approach*. Erlbaum Associates Publishers, 01 2005.

[23] J. Romano, J. Kromrey, J. Coraggio, and J. Skowronek, "Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen'sd for evaluating group differences on the NSSE and other surveys?" in *annual meeting of the Florida Association of Institutional Research*, 2006, pp. 1–3.

[24] Y. Benjamini and Y. Hochberg, "Controlling the false discovery rate: A practical and powerful approach to multiple testing," *Journal of the Royal Statistical Society*, vol. 57, no. 1, p. 289âĂŞ300, 1995.

[25] D. Talby, A. Keren, O. Hazzan, and Y. Dubinsky, "Agile software testing in a large-scale project," *IEEE Software*, vol. 23, no. 4, pp. 30–37, July 2006.

[26] F. Palomba, G. Bavota, M. Di Penta, R. Oliveto, D. Poshyvanyk, and A. De Lucia, "Mining version histories for detecting code smells," *Transactions on Software Engineering*, vol. 41, no. 5, pp. 462–489, 2015.

[27] B. Vasilescu, V. Filkov, and A. Serebrenik, "Perceptions of diversity on GitHub: A user survey," in *CHASE*, 2015, pp. 50–56.

[28] D. S. Cruzes and T. Dyba, "Recommended Steps for Thematic Synthesis in Software Engineering," in *ESEM*, 2011, pp. 275–284.

[29] J. A. Espinosa, S. Slaughter, R. E. Kraut, and J. D. Herbsleb, "Team knowledge and coordination in geographically distributed software development," *Journal of Management Information Systems*, vol. 24, pp. 135–169, 2007.

[30] P. Setia, B. Rajagopalan, V. Sambamurthy, and R. Calantone, "How peripheral developers contribute to open-source software development," *Information Systems Research*, vol. 23, no. 1, pp. 144–163, Mar. 2012. [Online]. Available: http://dx.doi.org/10.1287/isre.1100.0311

[31] A. Terceiro, L. R. Rios, and C. Chavez, "An empirical study on the structural complexity introduced by core and peripheral developers in free software projects," in *2010 Brazilian Symposium on Software Engineering*, 2010, pp. 21–29.

[32] F. Palomba, D. A. Tamburri, A. Serebrenik, A. Zaidman, F. Arcelli Fontana, and R. Oliveto, "Poster: How do community smells influence code smells," in *ICSE*, 2018.

[33] M. M. Lehman, "Programs, Life Cycles, and Laws of Software Evolution," *Proceedings of the IEEE*, vol. 68, no. 9, pp. 1060–1076, 1980.

[34] F. Ricca and A. Marchetto, "Are heroes common in FLOSS projects?" in *ESEM*, 2010, pp. 1–4.

[35] M. Torchiano, F. Ricca, and A. Marchetto, "Is my project's truck factor low?" in *WETSoM*, 2011, pp. 12–18.

[36] F. Ricca, A. Marchetto, and M. Torchiano, "On the difficulty of computing the truck factor," in *Product-Focused Software Process Improvement*. Springer, 2011, vol. 6759, pp. 337–351.

[37] C. Hannebauer and V. Gruhn, "Algorithmic complexity of the truck factor calculation," in *PROFES*, 2014, pp. 119–133.

[38] P. C. Rigby, Y. C. Zhu, S. M. Donadelli, and A. Mockus, "Quantifying and mitigating turnover-induced knowledge loss," in *ICSE*, 2016, pp. 1006–1016.

[39] M. Nassif and M. P. Robillard, "Revisiting turnover-induced knowledge loss in software projects," in *ICSME*, 2017, pp. 261–272.

[40] M. Hilton and A. Begel, "A study of the organizational dynamics of software teams," in *ICSE-SEIP*, 2018, pp. 1–10.

[41] B. Vasilescu, D. Posnett, B. Ray, M. G. J. van den Brand, A. Serebrenik, P. T. Devanbu, and V. Filkov, "Gender and tenure diversity in github teams," in *CHI*, 2015, pp. 3789–3798.

[42] H. S. Qiu, A. Nolte, A. Brown, A. Serebrenik, and B. Vasilescu, "Going farther together: the impact of social capital on sustained participation in open source," in *ICSE*, G. Mussbacher, J. M. Atlee, and T. Bultan, Eds. IEEE / ACM, 2019, pp. 688–699.

[43] C. Miller, D. G. Widder, C. Kästner, and B. Vasilescu, "Why do people give up flossing? A study of contributor disengagement in open source," in *OSS*, ser. IFIP Advances in Information and Communication Technology, F. Bordeleau, A. Sillitti, P. Meirelles, and V. Lenarduzzi, Eds., vol. 556. Springer, 2019, pp. 116–129.

[44] G. Iaffaldano, I. Steinmacher, F. Calefato, M. A. Gerosa, and F. Lanubile, "Why do developers take breaks from contributing to OSS projects? A preliminary analysis," *CoRR*, vol. abs/1903.09528, 2019. [Online]. Available: http://arxiv.org/abs/1903.09528

[45] A. Lee, J. C. Carver, and A. Bosu, "Understanding the Impressions, Motivations, and Barriers of One Time Code Contributors to FLOSS Projects: A Survey," in *ICSE*, 2017, pp. 187–197.

[46] G. Pinto, I. Steinmacher, and M. A. Gerosa, "More Common Than You Think: An In-depth Study of Casual Contributors," in *SANER*, 2016, pp. 112–123.

[47] I. Steinmacher, T. Conte, M. A. Gerosa, and D. Redmiles, "Social Barriers Faced by Newcomers Placing Their First Contribution in Open Source Software Projects," in *CSCW*, 2015, pp. 1379–1392.

[48] I. Steinmacher, T. U. Conte, C. Treude, and M. A. Gerosa, "Overcoming open source project entry barriers with a portal for newcomers," in *ICSE*, 2016, pp. 273–284.

[49] J. Coelho, M. T. Valente, L. L. Silva, and A. Hora, "Why We Engage in FLOSS: Answers from Core Developers," in *CHASE*, 2018.

[50] J. Coelho and M. T. Valente, "Why modern open source projects fail," in *FSE*, 2017, pp. 186–196.

[51] B. Adams, E. Constantinou, T. Mens, and G. Robles, Eds., *Proceedings of the 1st International Workshop on Software Health, SoHeal@ICSE 2018, Gothenburg, Sweden, May 27, 2018*. ACM, 2018. [Online]. Available: https://doi.org/10.1145/3194124

[52] P. Tourani, B. Adams, and A. Serebrenik, "Code of conduct in open source projects," in *SANER*, M. Pinzger, G. Bavota, and A. Marcus, Eds. IEEE Computer Society, 2017, pp. 24–33.

[53] D. Gachechiladze, F. Lanubile, N. Novielli, and A. Serebrenik, "Anger and its direction in collaborative software development," in *ICSE NIER*. IEEE Computer Society, 2017, pp. 11–14.