

Leaving Behind the Software History When Transitioning to Open Source: Reasons and Implications

Gustavo Pinto, Igor Steinmacher, Marco Gerosa

▶ **To cite this version:**

Gustavo Pinto, Igor Steinmacher, Marco Gerosa. Leaving Behind the Software History When Transitioning to Open Source: Reasons and Implications. 14th IFIP International Conference on Open Source Systems (OSS), Jun 2018, Athens, Greece. pp.50-60, 10.1007/978-3-319-92375-8_5 . hal-01875496

HAL Id: hal-01875496

https://inria.hal.science/hal-01875496

Submitted on 17 Sep 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

Leaving Behind the Software History When Transitioning to Open Source: Reasons and Implications

Gustavo Pinto[1], Igor Steinmacher[23] and Marco Gerosa[3]

[1] Federal University of Pará, Belém, PA, Brazil,
gpinto@ufpa.br
[2] Federal University of Technology, Paraná, Campo Mourão, Brazil,
igorfs@utfpr.edu.br
[3] Northern Arizona University, Flagstaff, EUA,
Marco.Gerosa@nau.edu

Abstract. Maintenance of software history is regarded to be one of the most relevant features of Version Control Systems (VCS) and is well-known to be indispensable for software developers. However, transitioning from proprietary to open source software poses a challenge: keeping the software history might make available years of historical records and internal matters from the company that built the software. On the other hand, removing the software history may disturb the development and may be harmful to new contributors. We conducted a survey with open source software projects that made this shift to investigate (1) the reasons why they removed the software history and (2) the challenges that developers face with the lack of availability of software history. Among the results, we found that the most common reason for removing the software history is because it is entangled with proprietary code (the fact that the history contains sensitive information appears next). Interestingly, most core developers believed that the lack of software history is, in the worst case, "*a very minor inconvenience.*"

1 Introduction

Maintaining software history, or commit history, is one of the main benefits of Version Controls Systems (VCSs). Developers refer to the software history not only when they need to navigate through changes related to their tasks, but also to learn from previous mistakes or to decide what to do next [23]. Indeed, a recent survey evidenced that software history is indispensable for developers: 61% of the respondents said to examine history up to a few times a day [4]. Practitioners also report acquiring knowledge when examining software history [14]. In particular, the recent introduction of social coding hosting websites made software history of open source software projects more accessible and understandable. As a consequence, even end users are taking advantage of the software history [11]. Researchers also leverage software history to conduct several studies, for instance, regarding the impact of co-changes on software maintenance activities [5], to estimate defects [20], or to fix bugs [3].

However, when open-sourcing their projects, several companies chose not to open the commit history. If decision makers decide to keep the software history with information from the entire development process, they might share sensitive data that is only supposed to be accessed by internal members of the software company (*e.g.,* database passwords). However, if decision makers decide not to keep the software history, they might introduce an additional burden for the software development team (*e.g.,* when dealing with maintenance tasks) or even for new contributors interested in joining the new open source project.

This kind of trade-off is relevant since many software companies, even those that were well-restrictive when it comes to publishing their software artifacts, are now releasing their former proprietary projects under open source licenses. As a notable example, in 2015, Apple released Swift, a programming language designed to be the successor of Objective-C on mobile platforms, under Apache 2, an open source software license. A quick search on HackerNews[4] — a developer-oriented news aggregator — reveals that there are more than 200 news regarding proprietary software projects that became open source[5].

To shed some light on the trade-offs of keeping the software history, we surveyed proprietary software projects that made the shift to open-source and did not open the software history. We selected 50 projects by searching at news aggregators, mailing lists, or README files (details in Section 2.2). To understand the reasons and challenges of working with a non-trivial, yet history-free open source project, we posted questions in the project issue trackers or mailing lists. Based on the responses, we characterized a variety of reasons that explain the lack of software history, such as: "[the history] contains sensitive information," "housekeeping needed," and "licensing and legal reasons." Still, regarding the challenges associated with the lack of software history, although some respondents acknowledge the importance of such history, they reported to have very few problems with it, as one maintainer mentioned, "*I'm probably the person most likely to access it, and I'd estimate that I use it only a few times per year*".

In summary, this paper makes the following main contributions:

- The reasons that lead project maintainers to leave behind the software history when transitioning to open source;
- The discovery that some project maintainers do not value software history as predicted by the literature.

2 Method

In this section, we describe our research questions (Section 2.1), the projects studied and how we found them (Section 2.2), and our survey design and application (Section 2.3).

[4] `https://news.ycombinator.com/`

[5] `https://hn.algolia.com/?query="isnowopensource"`, accessed on Jan 8th, 2018.

2.1 Research Questions

To guide our research, we investigated the following important but overlooked research questions:
RQ1: Why some projects do not open the software history when going open source?
Why: Although one might believe that the reasons behind the removal of the software history are straightforward (e.g., to protect sensitive data about the company), this research question is intended to bring evidence to confirm or refute this belief, as well as to uncover additional reasons.

RQ2: What are the challenges associated with the lack of software history?
Why: To understand the hidden challenges triggered by the lack of software history. This better understanding can, in turn, motivate researchers and tool makers to improve existing VCS tools in order to mitigate these challenges, or even to avoid the need of leaving behind the software history.

2.2 Studied projects

We selected a set of active and non-trivial proprietary software projects that recently (*i.e.*, no later than 2014) became open source. To identify these projects, we used a convenience sampling approach: we searched in mailing lists, blog posts, and newsletters for indication of whether a proprietary project became open source. We double checked the first commit(s) for anything indicating whether the source code was imported all at once or not. Such commit usually has an informative message and a high number of additions. For instance, the first commit from project `deepvariant` was named "Initial release of DeepVariant", and had 49,522 additions (0 deletions) in 270 files.[6] We started this search in June 2016 and proceeded until we found 50 instances of proprietary projects that deleted their software history when transitioning to open source. Among them were `IndexTank` from LinkedIn, `caravel` from Airbnb, `msbuild` from Microsoft, and `Haxl` from Facebook. Throughout this process, we found only eight projects that kept the software history. We sent open questions to the 50 selected projects and received answers from 35 projects, which had been considered for this study; the list of the 35 projects is available at Table 1. Although the list of studied projects is not exhaustive, it contains a variety of projects, with relation to their domains, programming language use, and size in terms of lines of code. Figure 1 depicts some characteristics of the studied projects.

2.3 Survey

To better understand the reasons for the removal and the problems related with the lack of history, we designed a survey aimed at gathering insights about the importance of the lack of software history. We asked four open questions:

[6] `https://github.com/google/deepvariant/commit/8b84eab`

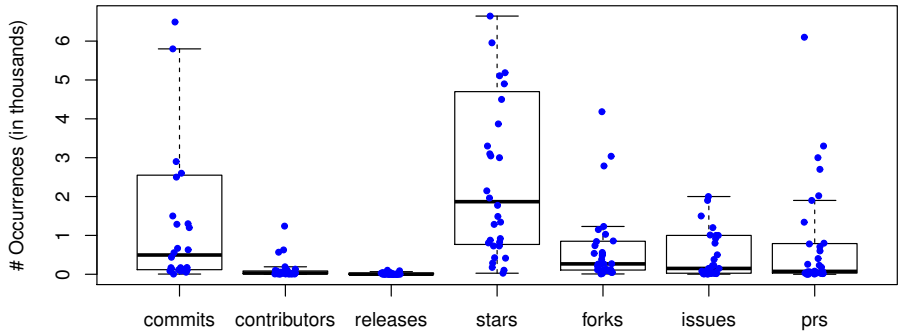


Fig. 1. Characteristics of the analyzed projects

Q1. Why did the software development team decide not to keep the software history?
Q2. Did the *core developers* face any kind of problems when trying to refer to the old history? If so, how did they solve these problems?
Q3. Did the *newcomers* face any kind of problems when trying to refer to the old history? If so, how did they solve these problems?
Q4. How does the lack of software history impact understanding and evolution of software?

We sent the questionnaire by creating issues in the issue trackers of the selected projects. This approach has been successfully employed in recent related work (e.g., [2]). Projects `Tensorflow` and `Scratch` closed the issues, suggesting other means to send research inquiries.[7][8] We, therefore, approached these projects through their mailing lists (`tensorflow` indeed replied our research inquiries through the mailing list: `http://bit.ly/2qR3Mm1`). When doing so, one mailing list user contacted us, asking: "*If you'd like another proprietary project that went open source and left behind its history, card.io did that,*" which we promptly accepted. In total, we collect answers from developers of 35 open source projects (totalizing 41 answers; we received up to four answers per project). For the majority of projects (project `ChakraCore` is the only exception), the respondents were within the top-10 most active ones.

To compile the survey results, we qualitatively analyzed the answers following coding procedures [22]. The qualitative analysis was conducted independently by the first two authors, followed by a consensus meeting. To enrich some of the findings, we opted to use some quotes throughout the results section. Among similar opinions, we chose to quote only the one we considered the most representative for each case.

[7] `https://github.com/LLK/scratch-flash/issues/1112`
[8] `https://github.com/tensorflow/tensorflow/issues/2361`

Table 1. The list of projects studied. Project `card.io` has no issue id because the maintainer personally got in touch with us by email. To see the response online, one needs to concatenate the project's URL + /issues/ + the issue id, resulting in, for instance, `github.com/Microsoft/msbuild/issues/621`.

Projects	URL	Issue Id
msbuild	github.com/Microsoft/msbuild	621
ChakraCore	github.com/Microsoft/ChakraCore	1280
heron	github.com/twitter/heron	1018
caravel	github.com/airbnb/superset	470
fbctf	github.com/facebook/fbctf	49
Tensorflow	github.com/tensorflow/tensorflow	2361
jsaction	github.com/google/jsaction	11
card.io	github.com/card-io/card.io-dmz	—
openwebrtc	github.com/EricssonResearch/openwebrtc	611
manta	github.com/joyent/manta	14
triton	github.com/joyent/triton	202
Dshell	github.com/USArmyResearchLab/Dshell	87
buffer-ios	github.com/bufferapp/buffer-ios-image-viewer	42
django-knowledge	github.com/zapier/django-knowledge	70
warp-ctc	github.com/baidu-research/warp-ctc	42
codecombat	github.com/codecombat/codecombat	3775
djinni	github.com/dropbox/djinni	253
superpowers-core	github.com/superpowers/superpowers-core	143
GameMaker	github.com/gandrewstone/GameMaker	2
opentoonz	github.com/opentoonz/opentoonz	640
magento2	github.com/magento/magento2	5654
IndexTank	github.com/linkedin/indextank-engine	43
ShareLatex	github.com/sharelatex/web-sharelatex	282
Haxl	github.com/facebook/Haxl	52
reason	github.com/facebook/reason	651
redex	github.com/facebook/redex	164
torchnet	github.com/torchnet/torchnet	28
torch	github.com/facebook/fb.resnet.torch	86
draft-js	github.com/facebook/draft-js	555
pinball	github.com/pinterest/pinball	74
decoda	github.com/unknownworlds/decoda	33
mrjob	github.com/Yelp/mrjob	1356
deepvariant	github.com/google/deepvariant	36
fsharp-support	github.com/JetBrains/fsharp-support	6
escape	github.com/ankyra/escape	4

3 RQ1: Why some projects do not open the software history?

While analyzing the answers, we observed that the lack of software history occurs for several reasons (some respondents described more than one reason):

Entangled with proprietary code (11 occurrences). We found that some projects became open source by open-sourcing a small part of a bigger project. As one respondent mentioned "*Extracting just the subfolder would have been difficult, and older versions would not have built.*" Another respondent summarized this process as: "*first get something working, and then disentangle it from your own proprietary code, configuration, etc.*" The same respondent also suggested that this might be a common pattern in OSS projects.

Contains sensitive information (11 occurrences). Some projects had hard-coded sensitive information (*e.g.*, credentials of a remote database) in the source code, e.g., "*DeepVariant was originally developed within Google, using our internal systems. [...] the earliest commits may contain information we cannot share, so upon releasing DeepVariant we squashed the history.*" Although one could simply delete the commits that alter this information, one respondent mentioned that they "*have to do an audit of the change descriptions to make sure the descriptions are appropriate for being published publicly,*" which they were unwilling to do. As another respondent said: "*Going through potentially thousands of commits, realistically, means no one will take on the heroic task of even open-sourcing the product.*" Thus, the only way to effectively remove this sensitive information is removing the entire software history.

Housekeeping needed (7 occurrences). As Fogel already anticipated [8], prior to releasing a proprietary software as open source, one needs to exert some effort toward improving code quality and documentation, as stated by one respondent: "*We cleaned out embarrassing or inappropriate comments, brought the code up to OSS standards, and generally improved code hygiene, robustness, and security.*" However, it was unexpected that the amount of refactoring required would prevent the software history from being useful, as another respondent said: "*the amount of reorganization that happened *just before* open-sourcing meant that it would be harder to track the history than to just understand the current state.*"

Of less importance (4 occurrences). Contrary to recent studies (*e.g.,* [4]), some developers believed that the software history does not deserve such importance. One respondent suggested that "*two of the primary motivations for keeping history are egos and understanding bug fixes. [our project] was low on ego, and we were careful to comment non-trivial or subtle bug fixes, so those two historical artifacts weighed less heavily.*" Another respondent highlighted another aspect of this lack of importance: "*remember that often something started as one person's random weekend project. Keeping a pristine history might not have been a priority.*"

License and legal reasons (4 occurrences). We also found non-technical and legal reasons. As one respondent mentioned: "*[Deleting the software history] also made it much easier to get the lawyers at our parent company to agree to open source it–instead of having to review the entire history for safety, they could review just the current state.*" Another reason is that some now open source applications rely on proprietary code. Therefore, to maintain license compliance, the developers have to maintain that code internally. Ultimately, one respondent

summarized this reason: "*Legal and policy reasons created incentives to release less source.*"

Did not use a VCS before (3 occurrences). Some projects became open source shortly after their bootstrap. Thus, there was no need to use any VCS, as one respondent said, "*there simply was no formal software history kept, and direct cooperation between experienced developers was sufficient to develop it to a releasable state.*" For these projects, the first commit at GitHub was their first use of a version control system.

Used another VCS before (3 occurrences). One respondent said that "*Before moving to GitHub, TensorFlow was developed on a system other than Git, and transferring history was not straightforward.*" Moreover, the same respondent suggested that the effort needed for migrating the software history from one VCS to another does not outweigh the benefits of keeping it: "*[the] value [of keeping the software history] was at best unclear, so we didn't do it.*"

4 RQ2: What are the challenges associated with the lack of software history?

Most of the project maintainers reported that they had few problems with the lack of software history; as one respondent mentioned: "*none of the core developers has wanted or needed to go look back through the history.*" Another respondent stated that "*based on practical experience, a history [of] more than a year is used very rarely*", which might explain this behavior. Moreover, another respondent suggested that institutional knowledge, i.e., the combined knowledge of the many contributors to a project, can be an effective substitute for formal software history, for instance: "*communication between developers, documentation in and outside of the source code, and the easily understood idiomatic expressions of the Python language were/are sufficient to maintain project coherency.*"

Even though these comments are in sharp contradiction with recent related work [4], it does not suggest that the software history is unimportant. Indeed, some respondents acknowledged its importance (*e.g.,* "*We very much agree that the software history is extremely useful for developers*"). Still, ten respondents suggested that the original software history is internally maintained, as one respondent highlighted: "*we still use the non-git system internally and can refer to history if we need to.*" However, the respondents also suggested that the internal software history is not actively used, as one respondent indicated: "*The old history is still available internally. I'm probably the person most likely to access it, and I'd estimate that I use it only a few times per year.*" Nevertheless, maintaining two software histories for the same project might require additional effort, which can make it difficult to track down the origins of the code.

Similarly, we found that maintainers do not think that the lack of software history is a significant problem for newcomers; all respondents shared this belief. However, most of the respondents said that they are not *aware* of any problems. Some newcomers may have faced problems, but not reported them, or gave up contributing. Project maintainers often suggested different ways that newcomers

can mitigate this problem, as one respondent mentioned: "*they have the code to look at and also code which is non-obvious should anyway have comments, else they could ask people who worked on the code before it was released.*"

Finally, regarding software evolution, we found a remarkable uniformity among the respondents; all of them believed that the lack of software history does not greatly impact software evolution and understanding. In the worse case, one respondent characterized the lack of history "*as a very minor inconvenience.*" Even more interestingly, one respondent said that software history of an active project loses importance over time: "*For a fast-moving project, history from more than half a year ago is not particularly valuable for development.*"

Along with this line of thought, Codoban *et al.* [4] observed that software developers need better tools to visualize software history. We believe that this lack of tools to properly visualize software history might create the perception that it is "not particularly valuable for development." Ultimately, we observed that the burden the lack of history may cause is "*certainly not enough to outweigh the costs of making it public,*" which is particularly relevant to the software projects under study.

5 Implications & Limitations

In this section we discuss some implications of our findings, and state the limitations of this work.

5.1 Implications

Based on our findings, we discuss some implications for stakeholders. We observed that some respondents mentioned a lack of tools to ease the migration between version control systems, for instance: "*I think this one may have been moving between source code repository technologies (SVN to Git) and the tools did not work well enough.*" Researchers and tool builders can propose a new set of tools to better support the transition between VCSs. Additionally, we observed that a common way to open-source a software project is by extracting only a small part of an existing project. However, this activity might be extremely difficult to conduct without appropriated tool support. As a result, developers leave behind the software history. Researchers can explore better techniques to extract only parts (or features) of the software while keeping only the relevant software history. Since some respondents mentioned that old software history is not particularly valuable for development, VCS designers can also propose lightweight VCSs, in which only the N most recent changes are kept, where N is based on project's activity or user choice. Programming language designers can also introduce programming language constructs that keep track of the evolution of certain parts of the code.

5.2 Limitations

As any empirical study, this one has limitations and threats to validity. First, we selected our projects by searching blog posts, newsletters, and README files; the first author manually conducted this process. Due to the qualitative nature of this approach (and the timeliness of a proprietary project becoming open source), one could find different projects. Moreover, we used GitHub's issues to send out our questionnaires. Such public participation can also be a threat since anyone could answer our questions. To mitigate this threat, we verified whether the respondents were active project members. We found that the majority of the respondents were among the top-10 most active contributors (only one respondent was not in the top-10). However, nine respondents do not appear as a contributor of the studied project. This might happen because the software history was removed and their contributions might have been removed as well. By comparing the user's affiliation and the project's affiliation on GitHub, we confirmed the affiliation of five of them. The remaining four did not state any affiliation. Also, we observed that some open source projects do not use issues for discussions. We, therefore, got in touch through the mailing list. Still, we certainly did not discover all challenges and reasons behind the lack of software history. Replications are necessary to fully understand the phenomenon. To facilitate replication, we made available the list of studied projects and the responses received in our survey on the companion website: `http://bit.ly/dataset-oss2018`.

6 Related Work

Some studies focus specifically on GitHub's features, which allow developers to track activities and form detailed impressions of social and technical abilities [12,24]. Community size, interest, and activeness have also being explored [17,21]. Moreover, there is a recent growth of studies targeting proprietary projects under development in social coding websites. Kalliamvakou *et al.* [10] examined how proprietary software projects use GitHub. They found that these projects apply practices such as reduced communication, independent work, and self-organization. Some research has investigated how proprietary projects adopt OSS-related practices to mitigate challenges related to the lack of communication and awareness [18,19]. More recently, we conducted some studies to understand how the contributions from employees (the ones hired by a software company) differs from volunteers (the ones the contribute in their free time) [7,15]. Regarding licensing, some studies investigated licenses inconsistencies and violations [9,1,13], and other focus on license evolution [6,25,26].

The closest work is an in-depth investigation of the contribution characteristics of Company-Owned OSS projects that kept the software history [16]. However, to the best of our knowledge, there is no study targeting the impact of the lack of the software history on proprietary projects that made the shift to open source software.

7 Conclusion

In this paper, we challenge an important belief related to the importance of software history. After the identification of a set of projects that left behind their entire software history after transitioning to open source, we deployed a survey to better understand (1) the reasons that lead to this removal and (2) the hidden challenges that arose due to its lack. We found eight reasons that might justify the decision of removing the software history, such as "entangled with proprietary code," "housekeeping needed," and "license and legal reasons." More interestingly, however, is the fact that when asked whether the lack of software history might impact understanding and evolving of the software, some respondents believed that the lack of history does not place any significant burden on developers.

For future work, we plan to better understand the newcomers' perception of the lack of software history and contrast our results with the analysis of projects that kept the history when migrating to open source.

Acknowledgments

We thank our respondents and the reviewers. This work is supported by CNPq #406308/2016-0; PROPESP/UFPA; and FAPESP #2015/24527-3.

References

1. Daniel A Almeida, Gail C Murphy, Greg Wilson, and Mike Hoye. Do software developers understand open source licenses? In *ICPC 2017*, pages 1–11. IEEE Press, 2017.
2. Guilherme Avelino, Leonardo Passos, Andre Hora, and Marco Tulio Valente. A novel approach for estimating truck factors. In *ICPC 2016*, pages 1–10, 2016.
3. Adrian Bachmann, Christian Bird, Foyzur Rahman, Premkumar Devanbu, and Abraham Bernstein. The missing links: Bugs and bug-fix commits. In *FSE'10*, pages 97–106, 2010.
4. Mihai Codoban, Sruti Srinivasa Ragavan, Danny Dig, and Brian Bailey. Software history under the lens: A study on why and how developers examine it. In *ICSME 2015*, pages 1–10, 2015.
5. Marcos César de Oliveira, Rodrigo Bonifácio, Guilherme N. Ramos, and Márcio Ribeiro. Unveiling and reasoning about co-change dependencies. In *Modularity 2016*, pages 25–36, 2016.
6. Massimiliano Di Penta, Daniel M German, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. An exploratory study of the evolution of software licensing. In *ICSE 2010*, volume 1, pages 145–154. IEEE, 2010.
7. Luiz Felipe Dias, Igor Steinmacher, and Gustavo Pinto. Who drives company-owned oss projects: Employees or volunteers? In *V Workshop on Software Visualization, Evolution and Maintenance*, VEM, page 10, 2017.
8. Karl Fogel. *Producing Open Source Software: How to Run a Successful Free Software Project*. O'Reilly Media, first edition, Feb 2013.

9. Daniel M German, Massimiliano Di Penta, and Julius Davies. Understanding and auditing the licensing of open source software distributions. In *ICPC 2010*, pages 84–93. IEEE, 2010.
10. Eirini Kalliamvakou, Daniela Damian, Kelly Blincoe, Leif Singer, and Daniel M. German. Open source-style collaborative development practices in commercial projects using GitHub. In *ICSE 2015*, pages 574–585, 2015.
11. Sandeep K. Kuttal, Anita Sarma, and Gregg Rothermel. On the benefits of providing versioning support for end users: An empirical study. *ACM Trans Comput-Hum Interact*, 21(2):9:1–9:43, February 2014.
12. Jennifer Marlow, Laura Dabbish, and Jim Herbsleb. Impression formation in online peer production: Activity traces and personal profiles in GitHub. In *CSCW*, 2013.
13. Rômulo Manciola Meloca, Gustavo Pinto, Leonardo Pontes Baiser, Marco Mattos, Ivanilton Polato, Igor Scaliante Wiese, and Daniel M German. A study of non-approved open-source licenses. In *MSR 2018*. IEEE Press, 2018.
14. Raphael Pham, Leif Singer, Olga Liskin, Fernando Figueira Filho, and Kurt Schneider. Creating a shared understanding of testing culture on a social coding site. In *ICSE 2013*, pages 112–121, 2013.
15. Gustavo Pinto, Luiz Felipe Dias, and Igor Steinmacher. Who gets a patch accepted first? comparing the contributions of employees and volunteers. In *11th IEEE/ACM International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE@ICSE 2018, Gothenburg, Sweden, May, 2018*, 2018.
16. Gustavo Pinto, Igor Steinmacher, Luiz Felipe Dias, and Marco Gerosa. On the challenges of open-sourcing proprietary software projects. *Empirical Software Engineering*, Mar 2018.
17. Gustavo Pinto, Igor Steinmacher, and Marco A. Gerosa. More common than you think: An in-depth study of casual contributors. In *SANER 2016*, pages 112–123, 2016.
18. D. Riehle, J. Ellenberger, T. Menahem, B. Mikhailovski, Y. Natchetoi, B. Naveh, and T. Odenwald. Open collaboration within corporations using software forges. *IEEE Softw*, 26(2):52–58, March 2009.
19. Srinarayan Sharma, Vijayan Sugumaran, and Balaji Rajagopalan. A framework for creating hybrid-open source software communities. *Inf. Syst. J.*, 12(1):7–26, 2002.
20. Maximilian Steff and Barbara Russo. Co-evolution of logical couplings and commits for defect estimation. In *MSR'12*, pages 213–216, 2012.
21. Igor Steinmacher, Gustavo Pinto, Igor Wiese, and Marco A. Gerosa. Almost there: A study on quasi-contributors in open-source software projects. In *ICSE'18*, 2018.
22. Anselm Strauss and Juliet M. Corbin. *Basics of Qualitative Research : Techniques and Procedures for Developing Grounded Theory*. SAGE, 3rd edition, 2007.
23. Yida Tao, Yingnong Dang, Tao Xie, Dongmei Zhang, and Sunghun Kim. How do software engineers understand code changes?: An exploratory study in industry. In *FSE'12*, pages 51:1–51:11, 2012.
24. Jason Tsay, Laura Dabbish, and James Herbsleb. Influence of social and technical factors for evaluating contribution in GitHub. In *ICSE'14*, pages 356–366, 2014.
25. Christopher Vendome, Gabriele Bavota, Massimiliano Di Penta, Mario Linares-Vásquez, Daniel German, and Denys Poshyvanyk. License usage and changes: a large-scale study on github. *Empir Softw Eng*, pages 1–41, 2017.
26. Christopher Vendome, Mario Linares-Vásquez, Gabriele Bavota, Massimiliano Di Penta, Daniel German, and Denys Poshyvanyk. Machine learning-based detection of open source license exceptions. In *ICSE 2017*, pages 118–129, 2017.