Forks

Proseminar: Software Analytics

Niklas Britz

Advisors: Prof. Sven Apel,
Christof Tinnes

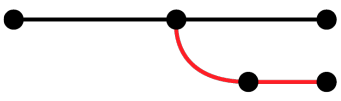
Saarland Informatics Campus,
Saarland University

Abstract. In Open Source Software, forking describes the act of creating a new repository by copying an old one.
This paper reflects on *state-of-the-art* research on forks. Perceptions and principles of forking have substantially changed over the last few decades. The developer community moved on from demonizing forks to the era of GitHub, where forking is not only pleased but encouraged. The paper discusses perceptions and stances towards forking and puts them into a historical context. It includes explanations about the motivation and processes behind forking. There is a distinction between *Hard Forks* and *Social Forks* (according to Zhou et al. [13]), which will be elaborated on in more detail. In addition, the paper contains information about the success (factors) of forks and communication, which are part of modern research in the field of forking.

1 Introduction

In Open Source Software, forking describes the act of creating a new repository by copying an old one. Those new repositories, so-called *forks*, can be copies of the original project or other forks. The process of forking a repository does not require permission from the owner in OSS. "The right to fork open source code is at the core of open source licensing" [7, p. 1].
The visualization of the repositories and the forking process looks like forking paths or a fork.

Fig. 1. Simple Fork



Forking developed from copying the source code manually to cloning the repository with the press of a button. The stance towards forking twisted from fear to encouragement. Not only did the perceptions around forking change, but also the conduction. Forking was primarily used to create a competing or superseding project [13]. A typical example of this type of fork would be *LibreOffice*, a fork from *OpenOffice.org*.
Today developers use forking as a tool of interaction. Developers propose changes or fix bugs through the creation of forks. This type of forking leads to greater success of the project [2]. Jiang et al. [5] have shown that the percentage of forked repositories had continuously increased before their research. That means that forking is a crucial topic in OSS. Hence, taking a closer look at the subject is very important.

2 Motivation and Processes behind Forking

2.1 Why Do We Fork?

In order to dive deeper into forking, one must have a look at the motivations and reasons for the creation of forks that developers have. Jiang et al.[5], therefore, surveyed developers in two rounds.
In the first round, they picked 1,000 developers who had forked 30 repositories or more (GitHub). The researchers interviewed those developers via e-mail and asked them why they had forked in the past. In the first round, the developers answered open-ended. Most of them argued that they fork in order to submit pull requests. That means their intention, after code changes had been made locally on the fork, is to submit those changes to the original repository. The owners of the *upstream* repository can then decide whether to accept or decline the change. Another significant portion of developers answered that they wanted to fix bugs and yet another wanted to add features. However, those categories do not imply reintegration. Around 9 % of developers stated that they fork to keep copies. The accurate figures can be seen in Table 1.
The researchers surveyed 3,000 developers who met the above criteria in the second round. Here, the developers had to answer multiple-choice questions.

Cause	Developers in the first round	Developers in the second round
Submit pull requests	57 / 46 %	128 / 79 %
Fix bugs	45 / 36.3 %	125 / 77.2 %
Add features	27 / 21.8 %	112 / 69.1 %
Keeping copies	57 / 46 %	128 / 79 %
Other	N/A	8 / 4.9 %

Table 1. Root causes of forking. Taken from [5, p. 554]

That explains the deviation from the first and second-round results. It explains why the relative percentage of developers in each category is higher than in the first round. The order of the root causes compared by their likeliness is still maintained (Table 1).

2.2 How Do Developers Select Repositories?

Furthermore, the researchers Jiang et al. [5] asked the developers how they selected the repositories they had forked. Just like the first question, the researchers surveyed in two rounds. Again, the first question was open-ended and the second one was multiple-choice. The results are displayed in Table 2. After evaluation, the team found out that most developers find a repository by clicking on external links. The websites developers mentioned most often were Twitter, Hacker News and Reddit. A few developers found projects on RubyGems. More than half of the developers found projects by searching for specific keywords and repositories matching their requirements. The search engines that the surveyees mentioned most often were Google and GitHub itself. Other ways of finding forkable repositories are recommendations (e.g., GitHub explore page) and friends whom the developers know (word of mouth, posts).

Mechanism	Developers in the first round	Developers in the second round
External links	72 / 58.1 %	107 / 66 %
Search	69 / 55.6 %	106 / 65.4 %
Friend	19 / 15.3 %	69 / 42.6 %
Recommendation	3 / 2.4 %	30/ 18.5 %
Other	N/A	25 / 15.4 %

Table 2. Mechanisms through which developers find and fork repositories. Taken from [5, p. 556]

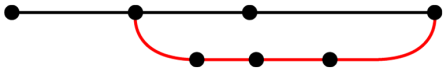
3 Social Forks

In their paper, Zhou et al. [13] distinguish between *Hard Forks* and *Social Forks*. They differ in their characteristics and implementation. First, *Social Forks* are discussed.

3.1 Social Forks: Implementing Features and Reintegrating

Social Forks are "often created for short-term feature implementation" [13, p. 446]. The researchers state that the intention when creating a *Social Fork*

Fig. 2. Example of a *Social Fork*. Black: Upstream repository. Red: Fork. Dots: Commits.



is to contribute the code changes back to the parent repository or to keep a copy. Usually, developers fork at a certain point, make some changes (implement features or fix bugs) and issue a pull request. The upstream repository owner can either accept or decline the change. *Social Forking* is a relatively young phenomenon that came up with the rise of GitHub. GitHub made it possible to fork instantly and to communicate efficiently.

3.2 Significant Communication between Repository and Forks

In their paper "We are Family: Analyzing Communication in GitHub Software Repositories and Their Forks", Brisson et al. [2] looked at how communication in software families works.
They defined a *software family* as a repository with its forks. The team analyzed 116,217,069 repositories they fetched from GHTorrent, a mirror of GitHub's REST API. They prepared the dataset by extracting software families that had been collaborative (participated in a significant amount of pull requests or issues), had had more than ten members and had been used for software development (e.g., not personal repository). Then they mined pull requests (PRs) and issues belonging to the respective repository. Additionally, they identified repository users who committed to the repository or merged with it.
The team found out that there is significant communication ($\hat{=}$ PRs and issues) within software family members.
They defined *PR from inside* as the "PRs that have occurred within the repository (via branching)" [2, p. 62]. At the same time, they defined *PR with repositories of other family members* as the "PRs that have occurred within the family (via forking)" [2, p. 62]. After analyzing their sample, the researchers discovered that 40.4 % of PRs were from inside and 59.6 % were with family members. So there is 'more' communication between the family members than in the repositories themself. Almost half of the users, the researchers identified, contributed to more than one repository in the respective families.

3.3 The Success of a Repository and the Role Communication Plays

Now that the researchers assessed quantitative communication features, they wondered how the communication affected the success of a repository. However,

how to measure success? Therefore, Brisson, Noei and Lyons [2] used the repository's star count to indicate its success. Users give a star to a repository to keep track of it, discover similar projects [1] or show appreciation and interest [2, p. 61].
To compare repository metrics and a repo's success, the researchers Brisson et al. [2] implemented a linear regression model with those metrics as input variables and the star count as the output variable. The metrics included figures like the number of forks, fork depth, pull requests from inside (from the repository via branching) or from the family (via forking), issues from users belonging to that repository or issues from users outside the family. Some metrics (those with Spearman's coefficient $|\rho| > 0.7$) were excluded from the linear regression model. "Including correlated variables in linear regression models negatively affects the stability of linear models, and hides the impact of each metric on the response variable" [8, as cited in [2]]. After the model was fitted and standardized, they analyzed the weights of the inputs. They then interpreted them as the respective metric's impact on the repository's star count (success). Some of the results were (statistically speaking!):

- The deeper the fork is in the 'forking tree', the fewer stars it has.
- The more forks a repository has, the more stars it has.
- The age of the repository has no significant impact on the star count.
- The more repository users contribute to other repositories in the same family, the more stars it has.
- The number of pull requests with family members (via forking) correlates significantly positive with the star count.
- However, the more the repository relies on pull requests from inside (via branching), the fewer stars it has.
- The number of created issues correlates positively with the star count (from users inside, outside **and** family).

Generally speaking, we can see that the more "communication" there is, the more successful a repository is. Communication between family members does influence a repository's popularity. The researchers went one step further.
They grouped each metric into four categories: non-communicative (e.g., age of repository), repository (e.g., number of PRs via branching), family (e.g., number of PRs via forking), outside (e.g., issues created from users outside the family). Then they analyzed how each category contributed to the model's fit by figuring out the relative importance of the inputs. Therefore they used the `pvmd`-score (proportional marginal variance decomposition) in the `relaimpo` `R`-package. Without going into detail: When trying to estimate how well the model explains the variance of a given dataset, one looks at the R^2-score.

$$R^2 = 1 - \frac{\text{Unexplained Variation}}{\text{Total Variation}} = 1 - \frac{\sum(y_i - \hat{y_i})^2}{\sum(y_i - \bar{y})^2}$$

Where y_i represents the output variable of an entry in the dataset. $\hat{y_i}$ is the estimation of the model for that entry and $\bar{y}$ is just the mean of the output

variable. Again, without going further into detail, the pvmd-score of an input variable is a sophisticated approach to quantify (in %) how much a variable influences the R^2 score [4] [12, p. 109]. The pvmd score of all variables must sum up to 100 %.
Brisson et al. [2] then found out that with 34.1 %, family communication influenced the model the most. Non-communicative metrics explained 30.3 % of the responsive variance. Metrics of the outside category contributed 22.7 % to the model's fit. The repository explained only 12.8 % of the fit. Hence, the researchers inferred that "Interactions involving family members contribute the most to repository star count." [2, p. 66].
We have now seen that creating forks on GitHub and communicating between the family members is highly beneficial and explanatory for a project's success. Due to this and because $> 99.9\%$ of GitHubs forks can be seen as *Social Forks* [13], we can clearly say that, from looking at the figures, *Social Forking* is very favorable. That is also what Zhou et al. [13] qualitatively found out in their interviews and research (from the developer's point of view). An accomplishment of GitHub.

4 Hard Forks

Now that *Social Forking* has been discussed and the characteristics, communication and success (factors) have been examined, we look at *Hard Forks*. In contrast to *Social Forks*, *Hard Forks* are a lot more controversial.

4.1 Hard Forks: A Competing Line of Development

Zhou et al. describe *Hard Forks* as forks that "continue a separate, often competing line of development" [13, p. 446]. The team that forks a project often wants to realize their vision of the project. A typical example of a *Hard Fork* is LibreOffice which was forked from OpenOffice.org in 2010.

4.2 Why to Create a Hard Fork?

There are many reasons for the creation of *Hard Forks*. In their paper, Robles and González-Barahona [10] list the most common reasons to create *Hard Forks*:

- **Technical reasons**: A part of the developer community wants to go down a different path than the repository owners. They want, for example, to include other functionality.
- **Governance reasons**: Developers might create a fork when they think feedback is not heard and the community is not considered.
- **Discontinuation of the original project**: The fork tries to revive a project that was discontinued in the past.

- **Commercial forks**: Sometimes, companies fork OSS to create their version of it and "meet some commercial interest"[10, p. 6]. It can also occur in the opposite direction. A community can create a *Hard Fork* from an OSS project associated with a company.
- **Personal reasons**: Interpersonal disputes and differences among the developer team on fundamental problems can trigger the creation of a *Hard Fork*.

By going over 220 different Wikipedia entries on *Hard Forks* Robles and González-Barahona figured out that the most common reasons for *Hard Forks* are are technical (27.3 %) followed by governance reasons (20.0 %). Zhou et al. [13] also stated that sometimes a *Social Fork* turns into a *Hard Fork* because, at some point, the repositories become too divergent and it is difficult to reintegrate.

4.3 What Types of Forks are There and How Do They Develop?

There are different types and outcomes of *Hard Forks*. First, we can distinguish between forks from active projects and forks that try to revive a dead project. As per Wheeler [11] and Robles and González-Barahona [10], there are five possible outcomes for forks:

- **Successful branching**: The original project and the fork are both successful and stay alive (with smaller communities each).
- **Fork merges back in the upstream repository**: "This is where the projects rejoin eachother" [11].
- **Discontinuation of the original project**: The fork supersedes the original project.
- **Discontinuation of the fork**: The fork is not successful.
- **Both fail**: Both projects fail after a particular time.

Zhou et al. [13] then tried to quantify those types and outcomes of *Hard Forks*. Therefore they constructed a classifier that classified whether repositories are *Hard Forks*. In a first step, they checked every repository that is a fork or has been forked on whether they meet one of seven heuristics (e.g., "fork of..." in the description, received external pull requests or have at least one year of development activity). When the repositories pass that first step, they are considered a candidate. Then the candidates were analyzed in detail by looking at the commit graphs and repository metadata.The researchers filtered out repositories used as course projects or storages, with less than three stars, without any commits after the fork, and those where 30 % or more of all commits were merged with the upstream repository. After classifying their labeled test set, they determined an accuracy of 95 % of the model, with few false negatives. After the researchers evaluated the model's output, they found out that *Hard Forks* are generally a rare phenomenon. Afterward, they classified 15 evolution patterns of *Hard Forks* like 'revives dead project and succeeds', 'revives dead project and does not succeed', 'forks an active project and both stay alive' or 'only merges'.

The researchers found out that 7 % of forks try to revive a dead project. Furthermore, out of the 93 % of forks that are forked from active projects, only 17 % show interaction with each other (merging, syncing). When they interviewed 18 developers (they selected them out of their classified *Hard Forks*) for 20 to 40 minutes each, they found out that many wanted to coordinate with the upstream repository owners. However, this conflicts with the figures mentioned above.

> *"What might explain this difference between intentions and observed actions is that synchronization and merging becomes difficult once two repositories diverge substantially and that monitoring repositories can becoming overwhelming with current tools"* [13, p. 453]

The researchers also found out that only 5 % of the time, both the upstream repository and the fork outlive for a more extended period. In 51 % of the cases, the *Hard Fork* lived longer than the upstream repository. 44 % of the time, the upstream repository lived longer than the fork.
Conclusively, we can say that *Hard Forks* are somewhat rare, but when they occur, there is mostly no interaction and infrequently do both projects stay alive at the same time.

4.4 Demonization of Hard Forks in the Past

Perceptions around forking have tremendously changed.
According to Zhou et al. [13], developers demonized forking in the 1990s and 2000s but also saw it as a fundamental right.
Developers thought that a fork, which back then was always understood as a *Hard Fork*, would fragment the developer community, duplicate effort and reduce communication [10].
Wheeler [11] argues that "those creating the fork are essentially stating that they believe the project's current leadership is ineffective, and are asking developers to vote against the project leadership by abandoning the original project and switching to their fork." He claims that the thread of a fork would be an encouragement for the repository owners to pay attention to needs and demands in the project's community.
Nevertheless, novel research indicates that this demonizing picture is deprecated and that the apprehensions of the developers do not hold water anymore.

4.5 How Hard Forks Influence the Sustainability of the Developer Community

Rastogi and Nagapan [9] investigated on how *Hard Forks* influence the sustainability of the developer community (on GitHub). They had, among other things, a look at what happened to the developer community after a fork occurred and how project characteristics influenced the outcome of a *Hard Fork*.
To measure how active a community is, the team looked at *developer community participation* (*dcp*), which they defined as the accumulated commit count of the

original project and the fork per time interval. They classified *Hard Forks* by selecting "Forks that do not send pull requests to predecessor forks, but observe internal commits" [9, p. 104], but called them independently developed forks.
Rastogi and Nagapan detected three cases of what could happen to the *dcp* after a fork occured. There could be an increase in *dcp*, a decrease in *dcp* or the *dcp* could stay the same.
First, they looked at internally developed projects on GitHub (not imported). They found out that, astonishingly, only 9 % of the time, the projects experienced a significant decrease in *dcp* and in 33 % of the cases, the *dcp* significantly increased. 58 % of the time, the dcp stayed more or less the same after the fork occurred. Then, the researchers had a look at projects that were imported to GitHub. However, the number of projects that experienced a decrease sharply rose to 20 %, while the portion of projects that observed an increase in *dcp* (30 %) and those that observed no significant change in *dcp* (50 %) decreased accordingly. The researchers have not provided any explanation as to why this is the case.

Rastogi and Nagapan [9] also examined how project characteristics at the time of forking influenced the *dcp* afterwards. A few things they found out were:

- The higher the **influence of the repository's owner**, the more likely the project is to maintain *dcp* after the fork.
  They measured the influence of the owner "in terms of the nature and size of the follower base" [9, p. 107]. It is given by the count of the follower's followers. That means that an owner is influential if he has many popular followers.
- The more **popular** (watchers' count) the upstream repository is, the lower the chances of decline in *dcp*.
- The **project's age** has a significant impact on the *dcp* after the fork occurs. In medium-sized projects (11-29 contributors), an increase of maturity (upstream repository) by one year reduces the odds declining *dcp* by 23 %. However, in large projects (> 30 contributors), the odds of decline rise by 18 %. The researchers leave no possible explanation for the divergence.

The research shows that the traditional perception of forks requires an overhaul. Rastogi and Nagapan showed that forking "increases developer community participation in more projects than it decreases developer community participation." [9, p. 111]. Nevertheless, the decrease influences a significant portion of projects [9].
This shows that further research on the impact forks have on the sustainability of the developer community is sensible and of immense interest.

4.6 (A)typical example: LibreOffice

In this section, the general findings examined above are applied to a concrete example: LibreOffice. LibreOffice (LO) is a well-known fork that has more than 10

years of active contributions, has significant commercial interest and has been adopted for professional use [3]. LibreOffice can as easily be described as the "Microsoft Office of Open Source".
It was forked from OpenOffice.org (OO) in 2010, after Sun Microsystems, which managed OO, was taken over by Oracle. The contributors were dissatisfied with Oracle's stance towards Open Source Software [3] [6]. Therefore, some influential committers forked LO from OO to realize their own vision. Hence, LibreOffice satisfies the characteristics of a hard fork.
What was the outcome? OpenOffice.org was continued by Apache under Apache OpenOffice (AOO). Although it still exists, LibreOffice outpaced AOO. It is one of the few times when both upstream repository and fork stay alive simultaneously [13]. However, LibreOffice is the project with higher community activity. In their study, Gamalielsson and Lundell [3] showed that LO had three to five times more monthly commits and committers in the years after LO was created By surveying OO and LO contributors, the researchers found out that many developers were frustrated with the license of OO and argued that "'it stopped being an Open Source project under Oracle'" [3, p. 138]. The surveyees argued that, when contributing to LO, they felt freed and enjoyed participating again. The team discovered that most of the contributors of OO then chose to contribute to LO.

The example of LibreOffice perfectly illustrates the findings of Rastogi and Nagapan [9]: Hard Forks cannot be demonized. As Gamalielsson and Lundell, when referring to LibreOffice, put it:

> *"It is thereby demonstrated that successful transfer and evolution of know-how and work practices can be achieved beyond individual Open Source software projects"* [3]

5 Conclusion

Forking is a core topic of Open Source development that is getting more and more popular. Especially in the past decade, the way researchers and the OSS community think about forks, has substantially changed.
Traditionally, forking referred to the creation of a copy of a repository that was then used to separate from the original project. Reasons for the creation of such can, for example, be of technical or personal nature or have something to do with the way a repository is governed [10]. Developers were afraid of them because they feared that the creation of those forks would fragment the developer community. Zhou et al.[13] called this type of fork *Hard Fork*. However, recent studies [9] [3] have shown that the demonization of *Hard Forks* is not sensible anymore. The sustainability of the developer community regarding *Hard Forks* depends on different factors. In this paper, the *Hard Fork* LibreOffice was discussed and compared with *Hard Forks* in general.
With the rise of GitHub, a new paradigm of forking emerged: *Social Forking*

[13]. *Social Forks* are used to fix bugs and add features, which then are reintegrated into the original project by submitting pull requests. Submitting pull requests is now the most common reason to create forks [**?**]. Communication between developers can happen via this process and it is shown that this type of communication benefits a project's success [2].

Conclusively, this paper has shown what forks are, how they work, how developers use them and how the perceptions and usages have changed over time.

References

1. Saving Repositories with Stars (2022), `https://docs.github.com/en/get-started/exploring-projects-on-github/saving-repositories-with-stars#about-stars`
2. Brisson, S., Noei, E., Lyons, K.: We Are Family: Analyzing Communication in GitHub Software Repositories and Their Forks. In: 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER). pp. 59–69 (2020)
3. Gamalielsson, J., Lundell, B.: Sustainability of Open Source software communities beyond a fork: How and why has the LibreOffice project evolved? Journal of Systems and Software 89, 128–145 (2014)
4. Groemping, U.: Relative Importance for Linear Regression in R: The Package relaimpo. Journal of Statistical Software 17(1), 1–27 (2006)
5. Jiang, J., Lo, D., He, J., Xia, X., Kochhar, P.S., Zhang, L.: Why and how developers fork what from whom in GitHub. Empirical Software Engineering 22(1), 547–578 (2017)
6. Noyes, K.: Don't Count on Oracle to Keep OpenOffice.org Alive (2010), `https://www.pcworld.com/article/502617/dont_count_on_oracle_to_keep_openoffice_org_alive.html`
7. Nyman, L., Lindman, J.: Code Forking, Governance, and Sustainability in Open Source Software. Technology Innovation Management Review 3, 7–12 (2013)
8. Rao, S.J.: Regression Modeling Strategies: With Applications to Linear Models, Logistic Regression, and Survival Analysis. Journal of the American Statistical Association 98(461), 257–258 (2003)
9. Rastogi, A., Nagappan, N.: Forking and the Sustainability of the Developer Community Participation – An Empirical Investigation on Outcomes and Reasons. In: 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER). vol. 1, pp. 102–111 (2016)
10. Robles, G., González-Barahona, J.M.: A Comprehensive Study of Software Forks: Dates, Reasons and Outcomes. In: Hammouda, I., Lundell, B., Mikkonen, T., Scacchi, W. (eds.) Open Source Systems: Long-Term Sustainability. pp. 1–14. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
11. Wheeler, D.A.: Why Open Source Software / Free Software (OSS/FS, FLOSS, or FOSS)? Look at the Numbers! (2015), `https://dwheeler.com/oss_fs_why.html`
12. Zhang, Z., Wang, L.: Advanced statistics using R. ISDSA Press, Granger, IN (2017)
13. Zhou, S., Vasilescu, B., Kästner, C.: How Has Forking Changed in the Last 20 Years? A Study of Hard Forks on GitHub. In: 2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE). pp. 445–456 (2020)