How Has Forking Changed in the Last 20 Years? A Study of Hard Forks on GitHub

Shurui Zhou
Carnegie Mellon University, USA

Bogdan Vasilescu
Carnegie Mellon University, USA

Christian Kästner
Carnegie Mellon University, USA

ABSTRACT

The notion of forking has changed with the rise of distributed version control systems and social coding environments, like GitHub. Traditionally forking refers to splitting off an independent development branch (which we call *hard forks*); research on hard forks, conducted mostly in pre-GitHub days showed that hard forks were often seen critical as they may fragment a community. Today, in social coding environments, open-source developers are encouraged to fork a project in order to contribute to the community (which we call *social forks*), which may have also influenced perceptions and practices around hard forks. To revisit hard forks, we identify, study, and classify 15,306 hard forks on GitHub and interview 18 owners of hard forks or forked repositories. We find that, among others, hard forks often evolve out of social forks rather than being planned deliberately and that perception about hard forks have indeed changed dramatically, seeing them often as a positive non-competitive alternative to the original project.

ACM Reference Format:
Shurui Zhou, Bogdan Vasilescu, and Christian Kästner. 2020. How Has Forking Changed in the Last 20 Years? A Study of Hard Forks on GitHub. In *42nd International Conference on Software Engineering (ICSE '20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3377811.3380412

1 INTRODUCTION

The notion of *forking* in open-source has evolved: Traditionally, forking was the practice of copying a repository and splitting off new independent development, often under a new name; forking was rare and was typically intended to compete with or supersede the original project [15, 30, 32]. Nowadays, forks in distributed version control systems are public copies of repositories in which developers can make changes, potentially, but not necessarily, with the intention of integrating those changes back into the original repository.

With the rise of social coding and explicit support in (distributed) version control systems, forking of repositories has been explicitly promoted by sites like GitHub, Bitbucket, and GitLab, and has indeed become very popular [19, 34]. For example, we identified over 114,120 GitHub projects with more than 50 forks, and over 9,164 projects with more than 500 forks as of June 2019, with numbers rising quickly. However, most of these modern forks are not forks in the traditional sense. As in our prior work [53], we distinguish between **social forks**, referring to creating a public copy of a repository on a social coding site like GitHub, often with the goal of contributing to the original project, and **hard forks**, referring to the traditional notion of splitting off a new development branch.

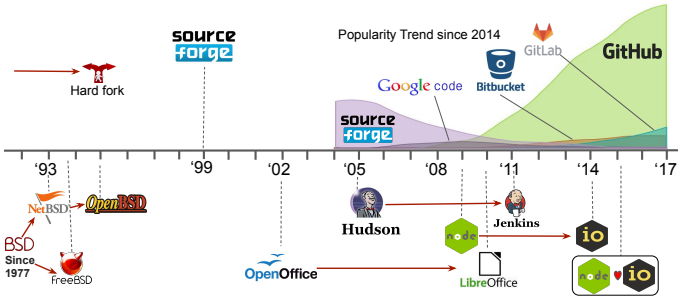


Figure 1: Timeline of some popular open-source forking events; popularity approximated with *Google Trends*.

Hard forks have been discussed controversially throughout the history of free and open-source software: On the one hand, free and open-source licenses codified the right to create hard forks, which was seen as essential for guaranteeing flexibility and fostering disruptive innovations [15, 30, 32] and useful for encouraging a survival-of-the-fittest model [48]. On the other hand, hard forks were frequently considered as risky to projects, since they could fragment a community and lead to confusion for both developers and users [15, 26, 30, 36], and there was a strong norm against forking; many well known hard forks exist (*e.g.*, *LibreOffice*, *Jenkins*, *io.js*; see Fig. 1), but there are not many well known cases where both communities survived and are both healthy after a hard fork, with a prominent exception being the BSD variants.

Prior research into forking of free and open-source projects focused on the motivations behind hard forks [8, 12, 13, 26, 31, 39, 47], the controversial perceptions around hard forks [6, 15, 26, 30, 36, 49], and the outcomes of hard forks (including studying factors that influence such outcomes) [39, 49]. However, essentially all that research has been conducted before the rise of social coding, much of it on SourceForge (GitHub was launched in 2008 and became the dominant open-source hosting site around 2012; cf. Fig 1).

In this paper, we argue that perceptions and practices around forking could have changed significantly since SourceForge's heydays. In contrast to the strong norm against forking back then, we conjecture that the promotion of social forks on sites like GitHub, and the often blurry line between social and hard forks, may have encouraged forking and lowered the bar also for hard forks. At the same time, advances in tooling, especially distributed version control systems like Git [7] and transparency mechanisms on social coding sites [10], may have enabled new opportunities and

ICSE '20, May 23–29, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7121-6/20/05.
https://doi.org/10.1145/3377811.3380412

changed common practices and perceptions. The professionalization of open-source development and the increasing involvement of corporations or even corporate ownership of open-source projects may have further tilted perceptions.

Therefore, we argue that it is time to revisit, replicate, and extend research on hard forks, asking the central question of this work: **How have perceptions and practices around hard forks changed?** Updating and deepening our understanding regarding practices and perceptions around hard forks can inform the design of better tools and management strategies to facilitate efficient collaboration. Furthermore, we attempt to automate the process of identifying hard forks among social forks and quantifying how frequent hard forks are across GitHub, which previous research did not cover.

Using a mixed-methods empirical design, combining repository mining with 18 developer interviews, we investigate:

- **Frequency of hard forks:** We attempt to quantify the frequency of hard forks among all the (mostly social) forks on GitHub. Specifically, we design and refine a classifier to automatically detect hard forks. We find 15,306 instances, showing that hard forks are a significant concern, even though their relative numbers are low.
- **Common evolution patterns of hard forks:** We classify the evolution of hard forks and their corresponding upstream repository to observe outcomes, including whether the fork and upstream repositories both sustain their activities and whether they synchronize their development. We develop our classification by visualizing and qualitatively analyzing evolution patterns (using card sorting) and subsequently automate the classification process to analyze all detected hard forks. We find that many hard forks are sustained for extended periods and a substantial number of hard forks still at least occasionally exchange commits with the upstream repository.
- **Perceptions of hard forks:** In interviews with 18 open-source maintainers of forks and corresponding upstream repositories, we solicit practices and perceptions regarding hard forks and analyze whether those align with ones reported in pre-social-coding research. We find that the 'stigma' often reported around hard forks is largely gone, indeed forks including hard forks are generally seen as a positive, with many hard forks complementing rather than competing with the upstream repository. Futhermore, with social forking encouraging forks as contribution mechanism, we find that many hard forks are not deliberately planned but evolve slowly from social forks.

Overall, we contribute (1) a method to identify hard forks, (2) a dataset of 15,306 hard forks on GitHub, (3) a classification and analysis of evolution patterns of hard forks, and (4) results from interviews with 18 open source developers about the reasons for hard forks, interactions across forks, and perceptions of hard forks.

Our research focuses on development practices on GitHub, which is by far the dominant open-source hosting platform (cf. Fig. 1) and has been key in establishing the social forking phenomenon. Even large projects primarily hosted on other sites often have a public mirror on GitHub, allowing us to gather a fairly representative picture of the entire open-source community. Our main research instruments are semi-structured interviews with open-ended questions and repository mining with GHTorrent [20] and the GitHub API. While our research is not planned as an exact replication of prior work and exceeds the scope of prior studies by comparing social and hard forks, many facets seek to replicate prior findings (*e.g.*, regarding motivations and outcomes of hard forks) and can be considered a conceptual replication [24, 43].

2 PAST RESEARCH ON FORKING

2.1 Types of Forking

What is popularly understood by 'forking a project' has changed in the last decades, which, in line with our prior work [53], we distinguish as *hard forks* and *social forks*:

- **Hard forks:** Traditionally, forking refers to copying a project in order to continue a separate, often competing line of development; the name and the direction of the project also typically change. Developers might fork a project, *e.g.*, when they are unhappy with the direction or governance, deciding to create a divergent version more in line with their own vision [15]. In pre-GitHub days, ways to contribute to an open-source project varied widely, but rather than using public forks one would typically create local copies to make changes and then send those as patch files.
- **Social forks:** Popularized through GitHub, 'forking' now also refers to public copies of open-source repositories that are often created for short-term feature implementation, often with the intention of contributing back the the upstream repository. A fork on GitHub is thus typically not intended to start an independent development line, but as a uniform mechanism for distributed development and third-party contribution (*i.e.*, pull requests) [10, 19]. In fact, the forking function on GitHub is frequently used even just as a bookmarking mechanism to keep a copy of a project without intention of performing any changes [25].

On GitHub, nowadays, both forms of forking exist, and we conjecture that the vast majority of forks are social forks, however it is not obvious how to distinguish the two kinds without a closer analysis.

At a technical level, forks can be created by cloning of repositories in distributed version control systems, in which case the fork maintains the history of the upstream project, or simply by copying files over and starting a new history (the latter was more common in pre-GitHub days). If forks are created directly on GitHub, a clone is automatically created, and GitHub tracks and visually shows the relationship between fork and upstream projects.

There is significant research on both hard forks and social forks. The hard-forking research is typically older, conducted almost exclusively before GitHub and social forking. Research on social forking is more recent, but focuses much more on the contribution process and issues around managing contributions in a single project.

2.2 Motivations for Forking

Reasons why developers might create a hard fork of an existing open-source project vary widely. Motivations for such forks have been studied primarily on *SourceForge* before the advent of social coding environment [8, 12, 13, 26, 31, 39, 47]. As per Robles and

González-Barahona [39], the most common motivations for hard forks were:

- *Technical.* Variants targeting specific needs or user segments that are not accommodated by the upstream project are the most common motivation [31]. As a project grows and matures, the contributors' goals or perspectives may diverge, and some may want to take the project in a different direction. If taken to the extreme, hard forks can be used for variant management, in which multiple related but different projects originating from the same source are maintained separately [3, 13, 14, 45].
- *Governance disputes.* Some contributors created hard forks when they feel their feedback is not heard or maintainers are accepting patches too slowly in the original project. A hard fork, or even just the threat of creating one, can help developers negotiate in governance disputes [17]; recent examples of hard forks caused by governance disputes include *Node.js* [42, 50] and *Docker* [51]. Other common forms of disputes occur when companies are involved and try to influence the direction of the project or try to close-source or monetize future versions of the project, as with *Hudson* and *OpenOffice*.
- *Discontinuation of the original project.* A hard fork can revive a project when the original developers have ceized to work on it. For example, back in the 1990s, the *Apache* web server project took over for the abandoned *NCSA HTTPd* project.
- *Commercial forks.* Companies sometimes fork open-source projects to create their own branded version of the project, sometimes enhanced with closed-source features. An example is Apple's fork of KDE's *KHTML* rendering engine as *Webkit*.
- *Legal reasons.* A project might consider different licenses, a trademark dispute may arise, or changes in laws (*e.g.*, regarding encryption) require technical changes. Hard forks can be used to split development for different jurisdictions.
- *Personal reasons.* Interpersonal disputes and irreconcilable differences of a non-technical nature lead to a rift between various parties, so the project forks. *OpenBSD* is a classic example.

In contrast to the older work on hard forks, more recent work has also investigated the motivation and practices behind social forks. For example, Fung et al. [16] report that only 14 percent of all active forks of nine popular JavaScript GitHub projects integrated back any changes. Subsequently, researchers studied social forks at larger scale and reported that around 50 percent of forks on GitHub never integrate code changes back [23, 53]. In addition, Jiang et al. [23] reported that 10 percent of their study participants used forks for backup purposes.

In our study, we revisit the question about the motivation for hard forks and explore whether they have changed with the rise of social coding.

2.3 Outcomes of Hard Forks

Wheeler [49] and Robles and González-Barahona [39] distinguish five possible outcomes of hard forks:

- *Successful branching, typically with differentiation.* Both the original project and the fork succeed and remain active for a prolonged period of time, fragmenting the community into smaller subcommunities. The *BSD* variants are notable examples.
- *Fork merges back into the upstream project.* The fork does not sustain independence but merges changes back into the upstream project, *e.g.*, after resolving a dispute that triggered the hard fork in the first place, as in the *io.js* fork of *Node.js* [50].
- *Discontinuation of the fork.* The fork is initially active, but does not sustain its activity. For example, when *libc* split off from *glibc*, the *glibc* maintainers invested in improvements to win back users and the fork failed.
- *Discontinuation of the upstream project.* The fork outperforms the upstream project such that the upstream project is discontinued (or the fork revives an already dead upstream project). For example, *XFree86* moved away from a GPL-compatible license, so the project forked and created *X.org*, which was quickly adopted by most developers and users; soon after, the *XFree86* core team disbanded and development ceased on the project.
- *Both fail.* Both projects fail (or the fork fails to revive a dead project).

Wheeler [49] conjectured that it is rare for both the fork and the upstream project to sustain activities. Robles and González-Barahona [39] quantified the frequency of each outcome in a sample of 220 forked open-source projects referenced from Wikipedia in 2011 (*i.e.*, selection biased toward well-known projects that have achieved a certain level of success) and found that successful branching was most common (43.6%), followed by discontinuation of the fork (29.8%) and discontinuation of the upstream project (13.8%); failure of both and merges were relatively rare (8.7% and 3.2%).

2.4 Pros and Cons of Hard Forks

Hard forks have long been discussed controversially. In the 90s and 2000s, forking was seen as an important right but also as something to avoid if at all possible, unless it is a last resort. There was a strong norm against forking, as it fragments communities and can cause hard feelings for the people involved. The free software movement has traditionally seen forking as something to avoid: forks split the community, introduce duplicate effort, reduce communication, and may produce incompatibilities [39]. Specifically, it can tear a community apart, meaning people in the community have to pick sides [6, 15, 26, 30, 36, 49]. Such fragmentation can also threaten the sustainability of open-source projects, as scarce resources are additionally scattered and changes need to be performed redundantly across multiple projects; *e.g.*, the 3D printer firmware *Marlin* fixed an issue (PR #10119) two years after the same problem was fixed in its hard fork *Ultimaker* (PR #118). At the same time, the right to forking is also seen as an important political tool of the community: The threat of a fork alone can cause project leaders to pay attention to issues they may ignore otherwise, should those issues actually be important and potentially improve their current practices [49].

In contrast, social forks are seen as something almost exclusively positive and are actively encouraged [4]. They are a mechanism to contribute to a project, and most open-source projects actively embrace external contributors [19, 46]. Although some maintainers complain about the burden of dealing with so many third-party contributions [21, 46] and some researchers warn about inefficiencies regarding lost contributions or duplicate work [38, 52, 53], we are not aware of any calls to constrain social forking.

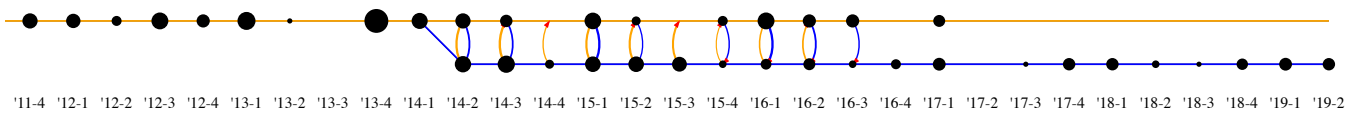


Figure 2: An example of commit history graph of fork `tmyroadctfig/jnode`

Importantly though, as our study will show, the distinction between social and hard forks is fluent. Social coding platforms contain both kinds of forks and they are not always easy to distinguish. Diffusion of efforts and fragmentation of communities, as always feared in discussions of hard forks, can be observed also on GitHub: Many secondary forks (*i.e.*, forks of forks) contribute to other forks, but not to the original repository, and forks slowly drift apart [16, 45]. A key question is, thus, whether the popularity of social forking encourages also hard forks and causes similar fragmentation and sustainability challenges feared in the past.

We believe it is necessary to revisit hard forking after the rise of social coding and GitHub. Specifically, we aim to understand the hard-fork phenomenon in a current social-forking environment, and understand how perceptions and practices may have changed.

3 RESEARCH QUESTIONS AND METHODS

As described in Sec. 2, the conventional use of the term *forking* as well as corresponding tooling have changed with the rise of distributed version control and social coding platforms, and we conjecture that this also influenced hard forks. Hence, our overall research question is **How have perceptions and practices around hard forks changed?**

We explore different facets of hard forks, including motivations, outcomes, and perceived stigma (cf. Sec. 2). We also attempt to identify how frequent hard forks are across GitHub, and discuss how developers navigate the tension and often blurry line between social and hard forks. We adopt a concurrent mixed-method *exploratory* research strategy [9], in which we combine *repository mining* – to identify hard forks and their outcomes – with *interviews* of maintainers of both forks and upstream projects – to explore motivations and perceptions. Mixing multiple methods allows us to explore the research question simultaneously from multiple facets and to triangulate some results. In addition, we use some results of repository mining to guide the selection of interviewees.

We explicitly decided against an exact replication [24, 43] of prior work, because contexts have changed significantly. Instead, we guide our research by previously explored facets of hard forks, revisit those as part of our repository mining and interviews, and contrast our findings with those reported in pre-GitHub studies. In addition, we do not limit our research to previously explored facets, but explicitly explore new facets, such as the tension between social and hard forks, that have emerged from technology changes or that we discovered in our interviews.

3.1 Instrument for Visualizing Fork Activities

We created *commit history graphs*, a custom visualization of commit activities in forks, as illustrated in Figure 2, to help develop and debug our classifiers (Sec. 3.2 and 3.3), but also to prepare for interviews. Given a pair of a fork and corresponding upstream repositories, we clone both and analyze the joint commit graph between the two, assigning every commit two one of five states: (1) *created before the forking point*, (2) *only upstream* (not synchronized), (3) *only in fork* (unmerged), (4) *created upstream but synchronized to the fork*, and (5) *created in the fork but merged into upstream*. Technically, in a nutshell, we build on our prior commit graph analysis [53], where merge edges are assigned weight 1 and all other edges weight 0, and the shortest path from the commit to any branch in either fork or upstream repository identifies where the commit originates and whether it has been merged (and in which direction).[1]

We subsequently plot activities in the two repositories over time, aggregated in three-month intervals; larger dots indicate more commits. In these plots, we include additional arrows for synchronization (from upstream into the fork) and merge (from fork to upstream) activities. With these plots, we can quickly visually inspect development activities before and after the forking point as well whether the fork and the upstream repository interact.

3.2 Identifying Hard Forks

Identifying hard forks reliably is challenging. Pre-GitHub work often used keyword searches in project descriptions, *e.g.*, 'software fork', or relied on external curated sources (*e.g.*, Wikipedia) [39]. Today, on sites like GitHub, hard forks use the same mechanisms as social forks without any explicit distinction.

Classifier development. For this work, we want to gather a large set of hard forks and even approximate the frequency of hard forks among all 47 million forks on GitHub. To that end, we need a scalable, automated classifier. We are not aware of any existing classifier except our own prior work [53], in which we classified forks as hard forks if they have at least two own pull requests or at least 100 own, unmerged commits and the project's name has been changed. Unfortunately, we found that this classifier missed many actual hard forks (false negatives), thus we went back to the drawing board to develop a new one.

We proceeded iteratively, repeatedly trying, validating, and combining various heuristics. That is, we would try a heuristic to detect hard forks and manually sample a significant number of classified forks to identify false positives and false negatives, revising the heuristic or combining it with other steps. Commit history graphs (cf. Sec. 3.1) and our qualitative analysis of forks (Sec 3.3 below) were useful debugging devices in the process. We iterated until we reached confidence in the results and a low rate of false positives.

[1] There are a few nuances in the process due to technicalities of Git and GitHub. For example, if the upstream repository deletes a branch after forking, the joint commit graph would identify the code as exclusive to the fork; to that end, we discard commits that are older than the forking timestamp on GitHub. Such details are available in our open-source implementation (https://github.com/shuiblue/VisualHardFork).

Our final classifier proceeds in two steps: first, we use multiple simple heuristics to identify *candidate* hard forks; second, we use a more detailed and more expensive analysis to decide which of those candidates are *actual* hard forks.

In the **first step**, we identify as candidate hard forks, among all repositories labeled as forks on GitHub, those that:

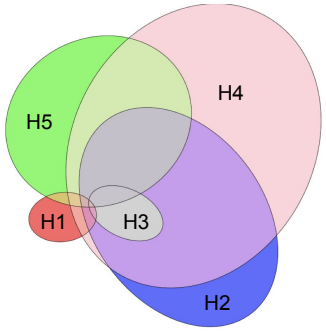
- *Contain the phrase "fork of" in their description* (H_1). We use GitHub's search API to find all repositories that contain the phrase "fork of" in their project description and are a fork of another project. The idea, inspired by prior work [31], is to look for projects that explicitly label themselves as forks (defined as "*self-proclaimed forks*"), *i.e.*, developers explicitly change their description after cloning the upstream repository. To work around GitHub's API search limit of 1000 results per query, we partitioned the query based on different time ranges in which the repository was created. Next, we compare the description of the fork and its upstream project to make sure the description is not copied from the upstream, i.e., that the upstream project is not already a self-proclaimed fork.
- *Received external pull requests* (H_2). Using the June 2019 GitHub dataset [18], we identified all GitHub repositories that are labeled as forks and have received at least *three* pull requests (excluding pull requests issued by the fork's owner to avoid counting developers who use a process with feature branches). We consider external contributions to a fork as a signal that the fork may have attracted its own community.
- *Have substantial unmerged changes* (H_3). Using the same GHTorrent dataset, we identify all forks that have at least 100 own commits, indicating significant development activities beyond what is typical for social forks.
- *Have at least 1-year of development activity* (H_4). Similar to the previous heuristic, we look for prolonged development activities beyond what is common for social forks. Specifically, we identify those forks as candidates in which the time between the first and the last commit spans more than one year.
- *Have changed their name* (H_5). We check if the fork's name in GitHub has been changed from the upstream repository's name (with Levenshtein distance ≥ 3). This heuristic comes from the observation that most social forks do not change names, but that forks intending to go in a different direction and create a separate community tend to change names more commonly (*e.g.*, *Jenkins* forked *Hudson*).

Each repository that meets *at least one* of these criteria is considered as a candidate. We show how many candidates each heuristic identified in the second column of Fig. 3b. Note, for all heuristics that use GHTorrent, we additionally validated the results by checking whether the fork and upstream pair still exist on GitHub and whether the measures align with those reported by the GitHub API.[2]

In the **second step**, we performed more detailed (and expensive) analyses of commit graphs and repository metadata in each candidate hard fork, to filter false positives:

- In line with prior work [25, 53], we remove repositories using GitHub for document storage or course project submission – some of which are among the most forked projects on GitHub. Specifically, after manual review, we discard repositories containing the keywords 'homework', 'assignments', 'course', 'codecamp', or 'documents' in their description; we discard repositories whose name starts with 'awesome-' (usually document collections); and we discard repositories with no programming-language-specific files (as per GitHub's language classification queried through the API).
- We discard candidates with fewer than three stars on GitHub. Stars are a lightweight mechanism for developers to indicate their interest in a project and a common measure of popularity. A threshold of three stars is very low, but still requires a minimum amount of public interest. According to GHTorrent data, of the 125 million GitHub repositories, 2 million repositories (1.6 %) have three or more stars.
- We discard candidates without any own commits after the fork, typically projects that only performed a name change as the single post-fork action.
- We discard candidates in which 30 % or more of all commits in the fork have been merged upstream, which indicates social forks with active contributions to the upstream project.
- For candidates identified with 100 commits or 1 year of activity, we discard those where the thresholds are not met when considering only unmerged commits exclusive to the fork.
- We discard candidates owned by developers who contributed more than 30 % of the commits or pull requests of the upstream repository, which typically indicates core team members of the upstream project using social forks for feature development.
- We discard candidates if the fork was created right after the upstream stopped updating if the fork is owned by an *organization* account and the upstream is owned by a *user* account. This is a common pattern we observed, indicating the ownership transfer.

Our classifier identifies a total of 15,306 hard forks across GitHub. In Fig. 3b, we show which heuristics identified the hard forks and the overlap between the different heuristics in Fig. 3a.



(a) Overlap between the heuristics.

Rule	Candidates	Actual
H_1	10,609	551
H_2	23,109	7,043
H_3	14,956	810
H_4	33,073	11,268
H_5	20,358	5,568
Total	63,314	15,306

(b) Hard forks identified.

Figure 3: Statistics on identified candidate hard forks and actual hard forks. Small intersections omitted for readability.

Classifier validation. To validate the precision of our classifier, we manually inspected a random sample of 300 detected hard forks. By manually analyzing the fork's and the upstream repository's history and commit messages, we classified 14 detected hard forks

[2]We include this step after identifying occasional errors in GHTorrent in our validation steps, such as switched fork-upstream relations between two repositories.

as likely false positives, suggesting an acceptable accuracy of 95 %. Note that manual labeling is a best effort approach as well, as the distinction between social and hard fork is not always clear (see also our discussion of interview results in Sec. 4.4).

Analyzing false negatives (recall) is challenging, because hard forks are rare, projects listed in previous papers are too old to detect in our GitHub dataset, and we are not aware of any other labeled dataset. We have manually curated a list of known hard forks from mentions in web resources and from mentions during our interviews. Of the 3 hard forks of which both the fork and the upstream repository are on GitHub, we detect all with our classifier, but the size of our labeled dataset is too small to make meaningful inferences about recall.

3.3 Classifying Evolution Patterns

We identified different evolution patterns among the analyzed forks using an iterative approach inspired by card sorting [44]. Evolution patterns describe how a hard fork and the corresponding upstream project coevolve and can help to characterize forking outcomes. In addition, we used evolution patterns to diversify interviewees.

Specifically, we printed cards with commit history graphs of 100 randomly selected hard forks (see Sec. 3.2), then all three authors jointly grouped the cards and identified a few common patterns. Our card-sorting was open, meaning we had no predefined groups; the groups emerged and evolved during the analysis process. Afterward, we manually built a classifier that detects the forks for each identified pattern. We then applied this classifier to the entire dataset, inspected that the automatically classified forks actually fit the patterns as intended (refining the classifier and its thresholds if needed). We then picked another 100 hard forks that fit none of the previously defined patterns and sorted those again, looking for additional patterns. We similarly proceeded within each pattern, looking at 100 hard forks to see whether we can further split the pattern. We repeated this process until we could not identify any further patterns.

After several iterations, we arrived at a stable list of 15 patterns with which we could classify 97.7 % of all hard forks. We list all patterns with a corresponding example commit history graph in Tab. 2. The patterns use characteristics that relate to previously found outcomes, such as fork or upstream being discontinued, but also consider additional characteristics corresponding to features that were not available or easily observable before distributed version control, *e.g.*, whether the fork and upstream merge or synchronize. We present the patterns in a hierarchical form, because our process revealed a classification with a fairly obvious tree structure, not because we were specifically looking for a hierarchical structure.

3.4 Interviews

To solicit views and perceptions, we conducted 18 semi-structured interviews with developers, typically 20 to 40 minutes. Despite reaching fewer developers, we opted for interviews rather than surveys due to the exploratory nature of our research: Interviews allow more in-depth exploration of emerging themes.

Interview protocol. We designed a protocol [2] that covers the relevant dimensions from earlier research and touches on expected changes, including reasons for forking, perceived stigma of forking, and the distinction and possible tensions between social and hard forks. We asked *fork owners* about their decision process that lead to the hard fork, their practices afterward (*e.g.*, why they renamed the projects), their current relationship to the upstream project (*e.g.*, whether they still monitor or even synchronize), and their future plans. In contrast, we asked *owners of upstream projects* to what extent they are aware of, interact with, or monitor hard forks; and to what degree they are concerned about such forks or even take steps to avoid them. In addition, we asked all participants with a long history of open-source activity if they observed any changes in their practices or perceptions and that of others over time.

Table 1: Background information of participants.

Par.	Domain	#Stars(U)	#Stars(F)	LOC	Role	Exp.(yr)
P1	Blockchain	<20	<10	10K	F	19
P2	Reinforcement learning	10K	1K	30K	F	3
P3	Mobile processing	-	70	20K	F	6
P4	Video recording	-	100	300K	F	18
P5	Helpdesk system	2K	<10>	800K	F	5
P6	CRM system	30	200	800K	F	10
P7	Physics engine	-	300	100K	F	15
P8	Social platform	500	230	500K	F	20
P9	Reinforcement learning	<20	<20	30K	2nd-F	3
P10	Game Engine	500	<10	200K	2nd-F	21
P11	Networking	300	100	500K	F	10
P12	Email library	-	10K	20K	F/U	32
P13	Game engine	3K	70	20K	F	11
P14	Machine learning	30K	50	60K	F	8
P15	Image editing	70	<10	20K	F	20
P16	Image editing	70	<10	20K	U	10
P17	Microcontrollers	9K	1K	300K	U	6
P18	Maps	400	<10	100K	U	9

F: Hard Fork Owner; U: Upstream Maintainer; 2nd-F: Fork of the Hard Fork
*Some of the upstream projects are not in GitHub, so the number of stars is unknown.
Numbers rounded to one significant digit.

All interviews were semi-structured, allowing for exploration of topics that were brought up by the participants. Our interview protocol evolved with each interview, as we reacted to confusion about questions and insights found in earlier interviews. That is, we refined and added questions to explore new insights in more detail in subsequent interviews – for example, after the first few interviews we added questions about the tradeoff between being inclusive to changes versus risking hard forks and questions regarding practices and tooling to coordinate across repositories. To ground each interview in concrete experience rather than vague generalizations, we focused each interview on a single repository in which the interviewee was involved, bringing questions back to that specific repository if the discussion became too generic.

Participant recruitment. We selected potential interviewees among the maintainers of the 15,306 identified hard forks and corresponding upstream repositories. We did consider maintainers with public email address on their GitHub profile that were active in the analyzed repositories within the last 2 years (to reduce the risk of misremembering). We sampled candidates from all evolution patterns (Sec. 3.3) and sent out 242 invitation emails.[3]

Overall, 18 maintainers volunteered to participate in our study (7 % response rate). Ten opted to be interviewed over email, one

[3]We unfortunately could not recruit interviewees in all roles for all patterns. For example, for 'reviving a dead project' we would not find any upstream maintainers that were active in the last 2 years.

through a chat app, and all others over phone or teleconferencing. In Table 2, we map our interviewees to the evolution pattern for the primary fork discussed (though interviewees may have multiple roles in different projects). Naturally, our interviewees are biased toward hard forks that are still active. Our response rate was also lower among maintainers of upstream repositories, who were maybe less invested in talking about forking. In Table 1, we list information about our interviewees and the primary hard fork we discussed. All interviewees are experienced open-source developers, specifically, many with more than 10 years experience of participating in open-source community, meaning they have interacted with earlier open-source platform such as *Sourceforge*. Our interviews reached saturation, in that the last interviews provided only marginal additional insights.

Analysis. We analyzed the interviews using standard qualitatively research methods [41]. After transcribing all interviews, two authors coded the interviews independently, then all authors subsequently discussed emerging topics and trends. Questions and disagreements were discussed and resolved together, if needed asking follow up questions to some interviewees.

3.5 Threats to Validity and Credibility

Our study exhibits the threats to validity and credibility that are typical and expected of this kind of exploratory interview studies and the used analysis of archival GitHub data.

Distinguishing between social and hard forks is difficult, even for human raters, as the distinction is primarily one of intention. In our experience, we can make a judgment call with high inter-rater reliability for most forks, but there are always some repositories that cannot be accurately classified without additional information. We build and evaluate our classifiers based on a best effort strategy, as discussed.

While we check later steps with data from the GitHub API, early steps to identify candidate hard forks may be affected by missing or incorrect data in the GHTorrent dataset. In addition, the history of Git repositories is not reliable, as timestamps may be incorrect and users can rewrite histories after the fact. In addition, merges are difficult to track if code changes are merged as a new commit or through 'squashing' rather than through a traditional merge commit. As a consequence, despite best efforts, there will be inaccuracies in our classification of hard forks and individual commits, which we expect will lead to some underreporting of hard forks and to some underreporting of merged code.

We analyze data with right-censored time series data, in which we can detect that projects have seized activity in the past, but cannot predict the future, thus seeing a larger chance for older forks to be discontinued.

Our study is limited to hard forks of which both fork and upstream repository are hosted on GitHub and of which the forking relationship is tracked by GitHub. While GitHub is by far the most dominant hosting service for open source, our study does not cover forks created of (typically older) projects hosted elsewhere and forks created by manually cloning or copying source code to a new repository. In addition, our interviews, as typical for all interview studies in our field, is biased toward answer from developers who chose to make their email public and chose to answer to our interview request, which underrepresented maintainers of upstream repositories in our sample.

4 RESULTS

We explore practices and perceptions around hard forks along four facets that emerged from our interviews and data.

4.1 Frequency of Hard Forks

Our classifier identified 15,306 hard forks, confirming that hard forks are generally a rare phenomenon. As of June 2019, GitHub tracks 47 million repositories that are marked as forks over 5 million distinct upstream repositories among GitHub's over 125 million repositories.

Among those, the vast majority of forks has no activity after the forking point and no stars. Most active forks have only very limited activity indicative of social forks. Only 0.2 % of GitHub's 47 million forks have 3 or more stars.

As our analysis of evolution patterns (Tab. 2) reveals, cases where both the upstream repository and the hard fork remain active for extended periods of time are not common (patterns 1, 2, and 4–7; 1157 hard forks, 8.8 %). Most hard forks actually survive the upstream project, if the upstream project was active when the fork was created (patterns 8–11; 7280 hard forks, 47.6 %), but many also run out of steam eventually (patterns 3 and 12–15; 6671 hard forks, 43.6 %).

While most hard forks are created as forks of active projects (patterns 4–15; 14254 hard forks, 93 %), there are a substantial number of cases where hard fork are created to revive a dead project (pattern 1–3; 1052 hard forks, 6.8 %), in some cases even triggering or coinciding with a revival of the upstream project (pattern 2; 56 hard forks, 0.36 %), but also here not all hard fork sustain activity (pattern 3; 420 hard forks, 2.7 %).

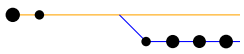
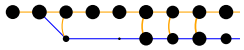
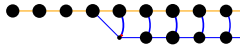
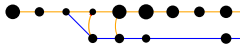
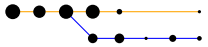
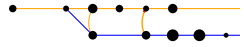
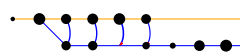
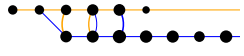
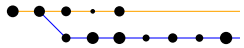
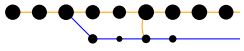
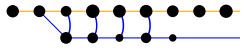
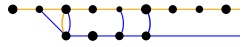
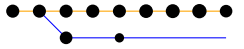
Discussion and implications. Even though the percentage of hard forks is low, the total number of attempted and sustained hard forks is not. Considering the significant cost a hard fork can put on a community through fragmentation, but also the potential power a community has through hard forks, we argue that hard forks are an important phenomenon to study even when they are comparably rare.

Whereas previous work typically looked at only a small number of hard forks, and research on tooling around hard-fork issues typically focus on few well known projects, such as the variants of *BSD* [35] or *Marlin* [28] or artificial or academic variants [14, 22], we have detected a significant number of hard forks, many of them recent, using many different languages, that are a rich pool for future research. We release the dataset of all hard forks with corresponding visualizations as dataset with this paper [2].

4.2 Why Hard Forks Are Created (And How to Avoid Them)

At a first glance, the interviewees give reasons for creating hard forks that align well with prior findings, including especially continuing discontinued projects or projects with unresponsive maintainers (P1, P2, P8), disagreements around project governance (P2, P12), and diverging technical goals or target populations (P3, P5,

Table 2: Evolution patterns of hard forks

Id		Category	Total	Sub-category	Example	Count	Interviewees
1	Revive Dead Project	Success (F. active > 2 Qt.)	632	Upstream remains inactive		576	P12
2				Upstream active again		56	
3		Not success (F active <= 2 Qt)	420			420	
4	Forking Active Project	Both Alive	723	only merge		26	P10
5				only sync		107	P2, P13, P15
6				merge & sync		28	P9
7				no interation		562	P1, P3, P4, P5, P7, P14
8		Fork Lived Longer	7280	only merge		174	
9				only sync		686	
10				merge & sync		107	
11				no interaction		6313	P6, P8, P11
12		Fork does not out live upstream	6251	only merge		388	
13				only sync		762	
14				merge & sync		199	
15				no interaction		4902	

P6, P11, P13, P14, P17). As discussed, we identified 1052 hard forks (Tab. 2, patterns 1–3, 6.8 %) that forked an inactive project.

An interesting common theme that emerged in our interviews though was that many hard forks were not deliberately created as hard forks initially. More than half of our interviewees described that they initially created a fork with the intention of contributing to the upstream repository (social fork), but when they faced obstacles they decided to continue on their own. Common obstacles were unresponsive maintainers (P1, P2, P8) and rejected pull requests (P11, P13, P14), typically because the change was considered beyond the scope of the project. For example, P2 described that "*before forking, we started by opening issues and pull requests, but there was a lack of response from their part. [We] got some news only 2 months after, when our fork was getting some interest from others.*" Similarly, some maintainers reported that a fork initially created for minor personal changes evolved into a hard fork as changes became more elaborate and others found them useful (P2, P14, P17); for example, P14 described that the upstream project had been constantly evolving and the code base became quickly incompatible with some libraries, so he decided to fix this issue while also adding functionality, after which more and more people found his fork and started to migrate.

Several maintainers also had explicit thoughts about how to avoid hard forks (both maintainers of projects that have been forked and fork owners who themselves may be forked), and they largely mirror common reasons for forking, i.e., transparent governance, being responsive, and being inclusive to feature requests. For example, P2 suggests that their project is reactive to the community, thus he considers it unlikely to be forked; similarly P16 decided to generally "*respond to issues in a timely manner and make a good*

faith effort to incorporate PRs and possibly fix issues and add features as the needs arrives" to reduce the need for hard forks. Beyond these, P2 also mentioned that they created a contributing guide and issue templates to coordinate with contributors more efficiently; P14 suggested to "*credit the contributors*" explicitly in release notes in order to keep contributors stay in the community.

Discussion and Implications. Whereas forking was typically seen as a deliberate decision in pre-GitHub days that required explicit steps to set up a repository for the fork and find a new name, nowadays many hard forks seem to happen without much initial deliberation. Social coding environments actively encourage forking as a contribution mechanism, which significantly lowers the bar to create a fork in the first place without having to think about a new name or potential consequences like fragmenting communities. Once the fork exists (initially created as social fork), there seems to be often a gradual development until developers explicitly consider their fork a separate development line. In fact, many hard forks seem to be triggered by rather small initial changes. These interview results align with the observation that only about 36 % of the detected hard forks on GitHub have changed the project's name (cf. Fig. 3a).[4]

More importantly, a theme emerged throughout our interviews that hard forks are not likely to be avoidable in general, because of a project's *tension between being specific and begin general*. On the one hand, projects that are more inclusive to all community contributions risk becoming so large and broad that they become expensive to maintain (e.g., as P17 suggests, the project maintainers need to take over maintenance of third-party contributions for niche use cases) and difficult to use (e.g., lots of configuration options and too much complexity). On the other hand, projects staying close to their original vision and keeping a narrow scope may remain more focused with a smaller and easier to maintain code base, but they risk alienating users who do not fit that original vision, who then may create hard forks. One could argue that hard forks are a good test bed for contributions that diverge from the original project despite their costs on the community: If fork dies it might suggest a lack of support and that it may have been a good decision not to integrate those contributions in the main project.

In this context, a family of related projects that serve slightly different needs or target populations but still coordinate may be a way to overcome this specificity-generality dilemma in supporting multiple projects that each are specific to a mission, but together target a significant number of uses cases. However, current technology does not support coordination across multiple hard forks well, as we discuss next.

4.3 Interactions between Fork and Upstream Repository

Many interviewees indicate that they are interested in coordinating across repositories, either for merging some or all changes back upstream eventually or to monitor activity in the upstream repository to incorporate select or all changes. Some hard fork owners did not see themselves competing with the upstream project, but rather being part of a larger project. For instance, although fork owner P13 has over 1500 commits ahead of the upstream project, he still said that "*I would not consider it independent because I am relying on what they (upstream) are doing. I could make it independent and stop getting their improvements, but it's to their credit they make it very easy for their many hundreds of developers to contribute patches and accept patches from each other. They regulate what goes into their project very well, and that makes [merging changes] into my fork much easier.*" Some (P4 and P11) indicate that they would like to merge, once the reason for the hard fork disappears (typically governance practices or personal disputes). Also upstream maintainers tend to be usually interested in what happens in their forks; for example, P17, a maintainer of a project with thousands of (mostly social) forks, said "*I try to be aware of the important forks and try to get to know the person who did the fork. I will follow their activities to some extent.*"

However, even though many interviewees expressed intentions, we see little evidence of actual synchronization or merging across forks in the repositories: For example, P1, P4, P8, and P11 mention that they are interested in eventually merging back with the upstream repository, but they have not done so yet and do not have any concrete plans at this point. Similarly, P2, P6, and P10 indicate that they are interested in changes in upstream projects, but do not actually monitor them and have not synchronized in a long time. Our evolution patterns similarly show that synchronization (from upstream to fork) and merging (from fork to upstream) are rare. Only 16.18 % of all hard forks with active upstream repositories ever synchronize or merge (Tab. 2, patterns 4–6, 8–10, and 12–14).

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. For example, P2 reports to only occasionally synchronize minor improvements, because the fork has diverged to much to synchronize larger changes; P10 has experienced problems of synchronizing too frequently and thus being faced with incomplete implementations and now only selectively synchronizes features of interest. In line with prior observations on monitoring change feeds [5, 10, 33, 52], interviewees report that systematically monitoring changes from other repositories is onerous and that current tools like GitHub's network graph are difficult to use and does not scale (P11, P16).

Discussion and Implications. Tooling has changed significantly since the pre-GitHub days of prior studies on hard forks which may allow new forms of collaboration across forks: Git specifically supports merges across distributed version histories, as well as selectively integrating changes through a 'cherry picking' feature. GitHub and similar social coding pages track forks, allowing developers to subscribe to changes in select repositories, and generally make changes in forks transparent [10, 11, 52]. Essentially all interviewees were familiar with GitHub's network view [1] that visually shows contributions over time across forks and branches.

Even though advances in tooling provide new opportunities for coordination across multiple forks and project maintainers are interested in coordinating and considering multiple forked projects as part of a larger community, current tools do not support this use case well. Current tools work well for short-term social forks but

[4] An interviewed hard-fork owners explained that they did *not* change the fork's name as a way to give credits to the upstream project, so not all hard forks without name changes should be automatically interpreted as being created through a gradual transition from social forks.

tend to work less well for coordinating changes across repositories that have diverged more significantly.

This provides opportunities for researchers to explore tooling concepts that can monitor, manage, and integrate changes across a family of hard forks. Recent academic tools for improved monitoring [33, 52] or cross-fork change migration [35, 37] are potentially promising but are not yet accessible easily to practitioners. Also more experimental ideas about virtual product-line platforms that unify development of multiple variants of a project [3, 14, 29, 40, 45] may provide inspiration for maintaining and coordinating hard forks, though they typically do not currently support the distributed nature of development with competing hard forks. A technical solution could solve the specificity-generality dilemma (cf. Sec. 4.2), allowing subcommunities to handle more specific features without overloading the upstream project and without fragmenting the overall community. We believe that our dataset of 15,306 hard forks can be useful to develop and evaluate such tools in a realistic setting.

4.4 Perceptions of Hard Forking

Our discussion with maintainers confirmed that the line between hard forks and social forks is somewhat subjective, but, when prompted, they could draw distinctions that largely mirror our definition (long-term focus, extensive changes, fork with own community). For example, P2 agree that his fork is independent from the upstream project because they have different goals, and suggests the fork has better code quality, and better community management practices; the only remaining connection are upstream bug fixes that he incorporates from time to time. Also, P6 considers his fork as independent, given a quicker release cycle and significantly refactorings to the code base.

For most interviewees, the dominant meaning of a fork is that of a social fork. When asked about perceptions of forks, most interviewees initially thought of social forks and have strong positive associations, e.g., others contributing to a project, onboarding newcomers and finding collaborators, and generally fostering innovation. For instance, P6 described the advantages of social forking as "*it encourages developers to go in a direction that the original project may not have gone,*" and similarly P9 thought that "*it could boost the creative ideas of the communities.*" One interviewee also mentioned that for young projects primarily focus on growth, having been forked is a positive signal meaning the project is useful to other people. Social forks were so dominant in the interviewees' mind as a default, that we had to frequently refocus the interview on hard forks. When asked specifically about hard forks, several interviewees raised concerns about potential community fragmentation (P4, P6, P17), worried about incompatibilities and especially confusing end users (P3, P9, P14, P17), and would have preferred to see hard-fork owners to contribute to the upstream project instead (P3, P8, P12). However, concerns were mostly phrased as hypotheticals and contrasted with positive aspects.

Many interviewed owners of hard forks do not see themselves competing with the upstream repository, as they consider that they address a different problem or target a different user population. For example, P10 described his fork as a *"light version"* of the upstream project targeting a different group of users.

While it is understandable that hard-fork owners see their forks as justified, also some interviewed owners of upstream projects had positive opinions about such forks. For example, P17 expressed that forks are good if there is a reason (such as a focus on a different target population, in this case beginners), and that those forks may benefit the larger community by bringing in more users to the project; P18 suggested even that he would support and contribute forks of his own project by occasionally contributing to them as long as it will benefit the larger community.

Discussion and Implications. Overall, we see that the perception of forking has significantly changed compared to perceptions reported in earlier work. Forking used to have a rather negative connotation in pre-GitHub days and was largely regarded as a last resort to be avoided to not fragment the community and confuse users. With GitHub's rebranding of the word *forking*, the stigma around hard forking seems to have mostly disappeared; the word has mostly positive connotations for developers, associated positively with external contributors and community. While there is still some concern about community fragmentation, it is rarely a concrete concern if there are actual reasons behind a hard fork. Transparent tooling seems to help with acceptance and with considering multiple hard forks as part of a larger community that can mutually benefit from each other.

We expect that a more favorable view, combined with lower technical barriers (Sec. 4.2) and higher expectations of coordination (Sec. 4.3) makes hard forks a phenomenon we should expect to see more of. However, positive expectations can turn into frustration (and disengagement of valuable contributors to sustain open source) if fragmentation leads to competition, confusion, and coordination breakdowns due to insufficient tooling.

With the right tooling for coordination and merging, we think hard forks can be a powerful tool for exploring new and larger ideas or testing whether there is sufficient support for features and ports for niche requirements or new target audiences (e.g., solving the specificity-generality dilemma discussed in Sec. 4.2 with a deliberate process). To that end though, it is necessary to explicitly understand (some) hard forks as part of a larger community around a project and possibly even explicitly encourage hard forks for specific explorations beyond the usual scope of social forks. We believe that there are many ways to support development with hard forks and to coordinate distributed developers beyond what social coding site offer at small scale today. Examples include (1) an early warning system that alerts upstream maintainers of emerging hard forks (e.g., external bots), which maintainers could use to encourage collaboration over competition and fragmentation if desired, (2) a way to declare the intention behind a fork (e.g., explicit GitHub support) and dashboard to show how multiple projects and important hard forks interrelate (e.g., pointing to hard forks that provide ports for specific operating systems), and (3) means to identify the essence of the novel contributions in forks (e.g., history slicing [27] or code summarization [52]).

5 CONCLUSION

With the rise of social coding and explicit support in distributed version control systems, forking of repositories has been explicitly promoted by sites like GitHub and has become very popular.

However, most of these modern forks are not hard forks in the traditional sense. In this paper, we automatically detected hard forks and their evolution patterns and interviewed open-source developers of forks and upstream repositories to study perceptions and practices. We found that perceptions and practices have indeed changed significantly: Among others, hard forks often evolve out of social forks rather than being planned deliberately and developers are less concerned about community fragmentation but frequently perceive hard forks a positive noncompetitive alternatives to the original projects. We also outlined challenges and suggested directions for future work.

Acknowledgements. Zhou and Kästner have been supported in part by the NSF (awards 1552944, 1717022, and 1813598) and AFRL and DARPA (FA8750-16-2-0042). Vasilescu has been supported in part by the NSF (awards 1717415 and 1901311) and the Alfred P. Sloan Foundation.

REFERENCES

[1] 2008. GitHub Network View. https://help.github.com/en/articles/viewing-a-repositorys-network.
[2] 2020. Appendix. https://github.com/shuiblue/ICSE20-hardfork-appendix.
[3] Michał Antkiewicz, Wenbin Ji, Thorsten Berger, Krzysztof Czarnecki, Thomas Schmorleiz, Ralf Lämmel, Ştefan Stănciulescu, Andrzej Wąsowski, and Ina Schaefer. 2014. Flexible Product Line Engineering with a Virtual Platform. In *Comp. Int'l Conf. Software Engineering (ICSE)*. ACM, 532–535.
[4] Matt Asay. 2014. Why you should fork your next open-source project. Blog Post. https://www.techrepublic.com/article/why-you-should-fork-your-next-open-source-project/
[5] Christopher Bogart, Christian Kästner, James Herbsleb, and Ferdian Thung. 2016. How to Break an API: Cost Negotiation and Community Values in Three Software Ecosystems. In *Proc. Int'l Symposium Foundations of Software Engineering (FSE)*. ACM, 109–120.
[6] Pete Bratach. 2017. Why Do Open Source Projects Fork? Blog Post. https://thenewstack.io/open-source-projects-fork/
[7] Caius Brindescu, Mihai Codoban, Sergii Shmarkatiuk, and Danny Dig. 2014. How Do Centralized and Distributed Version Control Systems Impact Software Changes?. In *Proc. Int'l Conf. Software Engineering (ICSE)*. ACM, 322–333.
[8] Bee Bee Chua. 2017. A Survey Paper on Open Source Forking Motivation Reasons and Challenges. In *21st Pacific Asia Conference on Information Systems (PACIS)*. 75.
[9] John W Creswell and J David Creswell. 2017. *Research design: Qualitative, quantitative, and mixed methods approaches*. Sage publications.
[10] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. 2012. Social coding in GitHub: transparency and collaboration in an open software repository. In *Proc. Conf. Computer Supported Cooperative Work (CSCW)*. ACM, 1277–1286.
[11] Laura Dabbish, Colleen Stuart, Jason Tsay, and James Herbsleb. 2013. Leveraging transparency. *IEEE Software* 30, 1 (2013), 37–43.
[12] James Dixon. 2009. Forking Protocol: Why, When, and How to Fork an Open Source Project. Blog Post. https://jamesdixon.wordpress.com/2009/05/13/different-kinds-of-open-source-forks-salad-dinner-and-fish/
[13] Neil A Ernst, Steve Easterbrook, and John Mylopoulos. 2010. Code forking in open-source software: a requirements perspective. *arXiv preprint arXiv:1004.2889* (2010).
[14] Stefan Fischer, Lukas Linsbauer, Roberto Erick Lopez-Herrejon, and Alexander Egyed. 2014. Enhancing clone-and-own with systematic reuse for developing software variants. In *Proc. Int'l Conf. Software Maintenance (ICSM)*. IEEE, 391–400.
[15] Karl Fogel. 2005. *Producing open source software: How to run a successful free software project*. O'Reilly Media, Inc.
[16] Kam Hay Fung, Aybüke Aurum, and David Tang. 2012. Social Forking in Open Source Software: An Empirical Study. In *Proc. Int'l Conf. Advanced Information Systems Engineering (CAiSE) Forum*. Citeseer, 50–57.
[17] Jonas Gamalielsson and Björn Lundell. 2014. Sustainability of Open Source Software Communities beyond a Fork: How and Why has the LibreOffice Project Evolved? *Journal of Systems and Software* 89 (2014), 128–145.
[18] Georgios Gousios. 2013. The GHTorent dataset and tool suite. In *Proc. Working Conf. Mining Software Repositories (MSR)*. IEEE Press, 233–236.
[19] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An exploratory study of the pull-based software development model. In *Proc. Int'l Conf. Software Engineering (ICSE)*. ACM, 345–355.
[20] Georgios Gousios, Bogdan Vasilescu, Alexander Serebrenik, and Andy Zaidman. 2014. Lean GHTorrent: GitHub data on demand. In *Proc. Working Conf. Mining Software Repositories (MSR)*. ACM, 384–387.
[21] Georgios Gousios, Andy Zaidman, Margaret-Anne Storey, and Arie Van Deursen. 2015. Work Practices and Challenges in Pull-Based Development: The Integrator's Perspective. In *Proc. Int'l Conf. Software Engineering (ICSE)*, Vol. 1. 358–368.
[22] Wenbin Ji, Thorsten Berger, Michal Antkiewicz, and Krzysztof Czarnecki. 2015. Maintaining Feature Traceability with Embedded Annotations. In *Proc. Int'l Software Product Line Conf. (SPLC)*. ACM, 61–70.
[23] Jing Jiang, David Lo, Jiahuan He, Xin Xia, Pavneet Singh Kochhar, and Li Zhang. 2017. Why and how developers fork what from whom in GitHub. *Empirical Software Engineering* 22, 1 (2017), 547–578.
[24] Natalia Juristo and Omar S Gómez. 2010. Replication of software engineering experiments. In *Empirical software engineering and verification*. Springer, 60–88.
[25] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M German, and Daniela Damian. 2016. An in-depth study of the promises and perils of mining GitHub. *Empirical Software Engineering* 21, 5 (2016), 2035–2071.
[26] Andrew M St Laurent. 2004. *Understanding Open Source and Free Software Licensing: Guide to Navigating Licensing Issues in Existing & New Software*. O'Reilly Media, Inc.
[27] Yi Li, Chenguang Zhu, Julia Rubin, and Marsha Chechik. 2017. Semantic slicing of software version histories. *IEEE Trans. Softw. Eng. (TSE)* 44, 2 (2017), 182–201.
[28] Max Lillack, Ştefan Stănciulescu, Wilhelm Hedman, Thorsten Berger, and Andrzej Wąsowski. 2019. Intention-based Integration of Software Variants. In *Proceedings of the 41st International Conference on Software Engineering (ICSE '19)*. IEEE Press, Piscataway, NJ, USA, 831–842.
[29] Leticia Montalvillo and Oscar Díaz. 2015. Tuning GitHub for SPL development: branching models & repository operations for product engineers. In *Proceedings of the 19th International Conference on Software Product Line*. ACM, 111–120.
[30] Linus Nyman. 2014. Hackers on forking. In *Proc. Int'l Symposium on Open Collaboration (OpenSym)*. ACM, 6.
[31] Linus Nyman and Tommi Mikkonen. 2011. To Fork or not to Fork: Fork Motivations in SourceForge Projects. In *Proc. IFIP Int'l Conf. on Open Source Systems*. Springer, 259–268.
[32] Linus Nyman, Tommi Mikkonen, Juho Lindman, and Martin Fougère. 2012. Perspectives on Code Forking and Sustainability in Open Source Software. *Open Source Systems: Long-Term Sustainability* (2012), 274–279.
[33] Rohan Padhye, Senthil Mani, and Vibha Singhal Sinha. 2014. NeedFeed: Taming Change Notifications by Modeling Code Relevance. In *Proc. Int'l Conf. Automated Software Engineering (ASE)*. ACM, 665–676.
[34] Ayushi Rastogi and Nachiappan Nagappan. 2016. Forking and the Sustainability of the Developer Community Participation–An Empirical Investigation on Outcomes and Reasons. In *Proc. Int'l Conf. Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 1. IEEE, 102–111.
[35] Baishakhi Ray, Miryung Kim, Suzette Person, and Neha Rungta. 2013. Detecting and characterizing semantic inconsistencies in ported code. In *Proc. Int'l Conf. Automated Software Engineering (ASE)*. IEEE, 367–377.
[36] Eric S Raymond. 2001. *The Cathedral & the Bazaar: Musings on linux and open source by an accidental revolutionary*. O'Reilly Media, Inc.
[37] Luyao Ren. 2019. Automated Patch Porting Across Forked Projects. In *Proc. Int'l Symposium Foundations of Software Engineering (FSE)*. ACM, New York, NY, USA, 1199–1201.
[38] Luyao Ren, Shurui Zhou, Christian Kästner, and Andrzej Wąsowski. 2019. Identifying Redundancies in Fork-based Development. In *Proc. Int'l Conf. Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 230–241.
[39] Gregorio Robles and Jesús M. González-Barahona. 2012. A Comprehensive Study of Software Forks: Dates, Reasons and Outcomes. In *Proc. IFIP Int'l Conf. on Open Source Systems*. 1–14.
[40] Julia Rubin and Marsha Chechik. 2013. A framework for managing cloned product variants. In *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 1233–1236.
[41] Johnny Saldaña. 2015. *The coding manual for qualitative researchers*. Sage.
[42] Anand Mani Sankar. 2015. Node.js vs io.js: Why the fork?!? Blog Post. http://anandmanisankar.com/posts/nodejs-iojs-why-the-fork/
[43] Stefan Schmidt. 2009. Shall we really do it again? The powerful concept of replication is neglected in the social sciences. *Review of General Psychology* 13, 2 (2009), 90–100.
[44] Donna Spencer. 2009. *Card sorting: Designing usable categories*. Rosenfeld Media.
[45] Stefan Stănciulescu, Thorsten Berger, Eric Walkingshaw, and Andrzej Wąsowski. 2016. Concepts, operations, and feasibility of a projection-based variation control system. In *Proc. Int'l Conf. Software Maintenance and Evolution (ICSME)*. IEEE, 323–333.
[46] Igor Steinmacher, Gustavo Pinto, Igor Scaliante Wiese, and Marco Aurélio Gerosa. 2018. Almost there: A study on quasi-contributors in open-source software projects. In *Proc. Int'l Conf. Software Engineering (ICSE)*. IEEE, 256–266.
[47] Robert Viseur. 2012. Forks impacts and motivations in free and open source projects. *International Journal of Advanced Computer Science and Applications* 3, 2 (2012), 117–122.
[48] Steve Weber. 2004. *The success of open source*. Harvard University Press.

[49] David A. Wheeler. 2015. Why Open Source Software/Free Software (OSS/FS, FLOSS, or FOSS)? Look at the Numbers! Blog Post. https://dwheeler.com/oss_fs_why.html
[50] Owen Willams. 2015. Node.js and io.js are settling their differences, merging back together. Blog Post. https://thenextweb.com/dd/2015/06/16/node-js-and-io-js-are-settling-their-differences-merging-back-together/
[51] Alex Williams and Joab Jackson. 2016. A Docker Fork: Talk of a Split Is Now on the Table. Blog Post. https://thenewstack.io/docker-fork-talk-split-now-table/
[52] Shurui Zhou, Ştefan Stănciulescu, Olaf Leßenich, Yingfei Xiong, Andrzej Wąsowski, and Christian Kästner. 2018. Identifying Features in Forks. In *Proc. Int'l Conf. Software Engineering (ICSE)*. ACM Press, 105–116.
[53] Shurui Zhou, Bogdan Vasilescu, and Christian Kästner. 2019. What the Fork: A Study of Inefficient and Efficient Forking Practices in Social Coding. In *Proc. Europ. Software Engineering Conf./Foundations of Software Engineering (ESEC/FSE)*. ACM Press, New York, NY, 350–361.