Hackers on Forking

Linus Nyman
Hanken School of Economics
Arkadiankatu 22
00100 Helsinki
+358 9 431 331
linus.nyman(at)hanken.fi

ABSTRACT

All open source licenses allow the copying of an existing body of code for use as the basis of a separate development project. This practice is commonly known as forking the code. This paper presents the results of a study in which 11 programmers were interviewed about their opinions on the right to fork and the impact of forking on open source software development.

The results show that there is a general consensus among programmers' views regarding both the favourable and unfavourable aspects that stem from the right to fork. Interestingly, while all programmers noted potential downsides to the right to fork, it was seen by all as an integral component of open source software, and a right that must not be infringed regardless of circumstance or outcome.

Categories and Subject Descriptors

K.6.3 [**Software management**]: *software development, software maintenance*

Keywords

Open source, free software, fork, code forking, code reuse

1. INTRODUCTION

All open source licenses permit the copying of an existing body of code to use as the basis for a separate development project. This practice is commonly known as code forking. The right to fork is a topic that has long been debated among practitioners and researchers [3] and the debate regarding the advantages and disadvantages of forking began even before the term "fork" came into popular use[1]. The practice of forking has been both complimented and condemned, described as a "Good Thing" [9], a "Bad Thing"[2], open source software's "cardinal sin" [25], and a vital part of open source software (e.g. [23]).

Code forking appears to be both loved and feared – the good, the bad, and the ugly all in one. Despite the conflicting and often strong views on forking, it is a topic that has seen little research. Indeed, Gamalielsson and Lundell [4] note that "there is clearly a need for increased knowledge about how OSS communities are affected by a fork."

Forking has the potential to affect open source software development and governance on many levels, including the software, community, and business ecosystem levels [14]. How programmer communities are affected by a fork will depend in no small part on how they view the fork. Thus, there would seem to be merit in gaining a better overall picture of how programmers[3] view forking. What do they view as the merits and drawbacks of forking? What makes a fork acceptable or unacceptable to the community? Given the strong opinions forking elicits, are there any moral obligations or codes of conduct that must be followed in order to gain community acceptance or favour when beginning a fork?

The aim of the current study was to better understand developers' views on forking, with an emphasis on what programmers consider the benefits and drawbacks of the right to fork, and what kinds of forks they consider favourable and unfavourable. How a fork is viewed by the wider open source community will directly affect its possibility of acquiring developers and attaining sustainability.

The paper is structured as follows: first, we cover previous discussions of, and research into, forking. Then we introduce the research questions and methods. After this the results of the study are presented. The paper concludes with a discussion of the implications of this research along with its limitations and some avenues for future research.

2. BACKGROUND

Code forking can be seen as healthy for the overall software ecosystem, offering a kind of 'survival of the fittest' where only the best code will survive [1, 23]. However, a fork is likely to be

OpenSym '14 , Aug 27-29 2014, Berlin, Germany
ACM 978-1-4503-3016-9/14/08.
http://dx.doi.org/10.1145/2641580.2641590

[1] See Nelson's (1993) discussion "Shattering – good or bad?": https://groups.google.com/forum/#!topic/gnu.misc.discuss/G7pwuPhnbEM

[2] See Jargon File: http://www.catb.org/jargon/html/F/forked.html

[3] The terms "programmer", "developer", and "hacker" are used interchangeably in this paper. The term hacker is thus not meant to imply any negative connotations (relating to exploiting weaknesses in a computer system or network, or indeed any other negative connotations).

harmful for the individual project affected [1, 23]. The harm is not due to the mere existence of a fork, but rather the loss of users, duplication of effort among programmers (e.g. [1, 11, 18]), and compatibility issues between forks ([21], see also [23]). A further disincentive is the challenge of attracting a sufficient pool of developers to aid in the further development of a fork, particularly if the original already has a significant community [23]. Programs under a copyleft license have further disincentives to forking: if a program is forked and improved, developers working on the original program may be able to incorporate those improvements, thus creating a disincentive to forking from both a development viewpoint (e.g. [9]), as well as a business-oriented viewpoint (e.g. [24]).

Thus there is a strong social pressure to avoid forking (e.g. [8, 18, 25]). Previous research has categorized the avoidance of forking a project as a community norm among open source software developers [22], and it has been noted that developers require a compelling reason to switch to a competing project [24].

Using a more restrictive definition of forking, it has previously been noted to be avoided. However, the importance of the right to fork is still underlined by many. The potential to fork a program has been called "the indispensable ingredient that binds developers together" [1], as since a fork is bad for everyone, the more serious the threat of a fork, the more willing people are to compromise in order to avoid it. Thus, the potential to fork is also a significant element in the governance of open source programs (e.g. [10, 23, 25]; see also [3, 4]), as no one has a "magical hold" [1] over any project since anyone can fork any project at any time. The right to fork can furthermore be seen as "a vital safety valve" [9] in case existing developers stop working on the project, or decide to stand in the way of progress. Similarly, regarding corporate actions and acquisitions, should a company try to "hijack" [6] the source code by making it proprietary, or otherwise work against the best interests of the program and its community, forking provides the community a way to protect their own vision of the project [10, 12, 14, 16, 17].

While forking is discussed in a number of textbooks and academic articles, there are few studies whose focus has been code forking itself. A small group of papers examine forking on a conceptual level, focusing on its relevance to the sustainability and governance of open source software development [14, 16, 17]. This author and Mikkonen [15] analysed the developer-stated motivations of forks hosted on the SourceForge repository, noting that the majority of forks appear to be motivated by pragmatic concerns, and to be of a non-competitive nature. Indeed, an in-depth analysis of 220 forks showed that forking: occurs in every software domain; has become more frequent in recent years; very few forks merge with the original project; and that many forks are of a "friendly" rather than competitive nature [20]. Competitive forks do exist, and may be motivated by a concern for the future wellbeing of the codebase under its present leadership [12].

Through the analysis of the 9 most forked JavaScript programs hosted on GitHub, Fung, Aurum, and Tang [2] introduced the concept of social forking. Social forking includes community, software, as well as other artefacts taken from the original project in the creation of the new. Gamalielsson and Lundell [3 and 4] analysed the LibreOffice fork from OpenOffice, finding that it is possible for a fork to prosper and achieve sustainability over time, and for a software community to outlive open source software projects.

While a fork has traditionally been defined as an instance in which a group of developers split into competing groups that go on to develop the program in different directions, Nyman [13] notes that the definition of a fork has changed in recent years, with several different interpretations in existence, but that still maintain the common thread of taking an existing program and modifying it. Furthermore, subsequent with the rise of popularity of hosting and revision control services with a broader use of the term[4], 'fork' has come to be used in an increasingly broad fashion. Indeed, in addition to being used by some as a synonym with branch, fork may also now be seen used among developers to indicate reusing existing code in the creation of a program that may target a significantly different user group than the original, developed by hackers with no affiliation with the original project.

3. RESEARCH QUESTIONS

The research questions focus on two broad areas of interest: programmers' views on the concept of forking in general and on the practice of forking. These two broad areas of interest can be divided into three topics:

1. What do programmers consider to be the benefits and drawback related to the right to fork? (Research questions 1 and 2)
2. What constitutes an acceptable and unacceptable fork? (Research questions 3 and 4)
3. How should forking be done, i.e., what, if any, are the moral obligations and proper codes of conduct in regards to forking? (Research questions 5 and 6)

3.1. What are the Benefits of the Right to Fork?

Existing literature has noted benefits like keeping a community more tightly knit, faster progress due to less divergent efforts, and the ability for the community to take control of their own fork of a project, should such a need arise. Are there any benefits developers consider significant that have escaped mention in previous literature?

3.2. What are the Drawbacks of the Right to Fork?

Generally, there is more focus on the negative aspects of forking, including issues such as redundancy of effort and confusion among users regarding which version to use. What do programmers see as the main drawbacks of the right to fork? Are there any perceived drawbacks not noted in previous literature?

3.3. When is it Acceptable to Fork?

Open source licenses guarantee the right to fork. Perhaps not surprising given its potential negative impact, several authors have noted the social pressure *not* to fork. However, previous research (e.g. [12, 15, 20]) has identified at least some circumstances in which a fork is considered acceptable. What kinds of circumstances are required for this social pressure not to fork to be outweighed by the perceived benefits *to* fork? Given that there are situations in which a community accepts or calls for a fork, can we identify any common characteristics of these situations?

[4] See GitHub: https://help.github.com/articles/fork-a-repo.

3.4. When is it Not Acceptable to Fork?

While several challenging aspects of forking have been noted, can one more precisely document what the conditions are under which the social pressure not to fork would be particularly strong? Are there any kinds of forks that are not considered acceptable under any circumstances?

3.5. Are There any Moral Obligations When Forking?

The large amount of forks documented in recent studies appears to contradict the notion of an open source norm not to fork. However, given the notion of the hacker ethic (e.g. [5, 7]; see also [18, 19]), it seems plausible that some norms surrounding a practice as pivotal as forking would exist, perhaps in the form of some unwritten yet commonly held rules or guidelines as to what obligations a programmer has, should they decide to fork a program. Do such guidelines exist? And if so, what are they?

3.6. What is the Proper Code of Conduct When Forking a Program?

Like the previous question, this question addresses a situation in which one has already decided to fork a program. How should one then proceed? Are there any undocumented rules or common practices regarding such a situation? If programmers could decide how other programmers went about forking, what kind of actions or behaviour would they hope for or expect?

4. RESEARCH APPROACH

4.1. Data and Data Gathering

Before beginning the data acquisition, it was important to become familiar with the writings and thoughts on forking found in academic literature, practitioner texts, and other sources of knowledge on forking (developer blogs, online news sites, etc.). Based on this knowledge, an interview guide consisting of open-ended questions was compiled. During the winter of 2012, a pilot study was conducted consisting of 4 semi-structured interviews. The interviewees all fulfilled the criteria that they all either were, or had previously been involved in open source software programming. During these interviews, the interview guide was tested and modified. The interviews were conducted in person, and were recorded and transcribed.

Eight additional interviews were conducted between 2013 and 2014. The interviewees were chosen using snowballing, with the criteria that: 1) they needed to have been involved in at least one open source software project; and 2) they needed to currently make a living from programming, be studying programming, or have at some point previously had made their living from programming. The interviews were conducted in Finnish, Swedish, and English, depending on the preference of the interviewee. Roughly half of the interviews were conducted face-to-face, the rest using Skype. The interviews were semi-structured, allowing for additional questions as needed, as well as giving the respondents the opportunity to express all thoughts and insights they felt relevant. All of the interviews were recorded and transcribed to aid in the data analysis. The shortest interview was 13 minutes and the longest was 41 minutes.

The extended time during which the interviews were conducted allowed for long periods of reflection of the results. The focus of the analysis was primarily to identify all unique opinions brought up in regard to each research question. The secondary goal was to examine these unique answers to see if they lent themselves to grouping. Where it seemed helpful, the answers were grouped into overarching themes. Upon completion of the first version of the article, it was sent to a subset of interviewees. This was done both to allow the interviewees to check their quotes, and to give them a chance to voice their opinions on the points brought up by other programmers.

A majority of the programmers were from Finland, the rest were from Italy, Russia, and the Ukraine. The interviewees ranged in age from their early twenties to their early fifties. One interviewee was female, the rest were male. The programmer with the least experience was involved in his first open source project at the time of interview. The most experienced programmers had several decades of experience, and included a developer who had worked primarily on the Linux kernel, a developer employed by the Apache Foundation, a developer who was a founding member of both MySQL and MariaDB, and a Linux pioneer who has furthermore both authored as well as contributed to numerous other open source programs.

4.2. Unifying the Definitions of "Fork"

Due to the potential for differences among interviewees regarding their views of what, exactly, the term fork entails [13], all interviewees were first asked a number of questions regarding their definition of the term fork and what it entailed. This was to ensure that all interviewees would be discussing a similar phenomena. Of all of the interviews, one programmer had a definition of fork that was unique enough that the answers given during that interview are not included in this study. Thus, the data for the study originally consisted of 12 interviews, of which 11 were included in the study. For the purposes of this paper, the definition of fork used is a somewhat broader version of Robles and González-Barahona's definition [20]. In this study a fork is when one or more developers start an independent line of development based on the source code of an existing project. Projects that will or may attempt to submit their changes back to the original are also included in the definition used herein.

5. FINDINGS

5.1. What are the Benefits of the Right to Fork?

When asked about the benefits of the right to fork, programmers commonly shared that forking is something more than a mere right guaranteed by open source. Instead, they saw it as an integral part of the very concept of open source software. As one programmer noted, "*I don't know if one really could see open source as open source without the right to fork*." Another programmer stated that if developers can't fork, "*then first of all, it isn't free software, it isn't open source anymore.*" One programmer discussed the topic in greater historical depth:

"At its root it is a large philosophical question: the whole idea of open source takes as its starting point that you can take source code and do whatever you want with it, entirely regardless of what the original developer or current community wants to do with the code. And in that sense it is one of the pillars of open source, this possibility to fork."

Thus, the first point to note is that the right to fork, regardless of any specific benefits it offers, is commonly seen as a cornerstone of open source.

Moving on to specific benefits brought up regarding the right to fork, one developer, with several decades experience in open source software development, noted that forking *"is what makes it* *possible to take a project in different directions or [to ensure] that the product doesn't die."* This developer also brought up a benefit for companies: that they can use the right to fork to solve problems through forking a program and then customizing it to meet their specific needs. If broadly interpreted, the three main points of taking a program in a different direction, keeping a program alive, and benefits for corporations – or, indeed, non-profits or any other interested party – cover the entire list of benefits stated by all interviewees combined. However, to better understand and convey the thoughts of the participants, it is useful to divide the stated benefits into further categories. The benefits brought up by the interviewees can thus be grouped into three overarching categories: preservation, experimentation, and customization.

Preservation

A rather clear-cut benefit is that forking makes it possible to continue a project that has been abandoned[5]. As a developer from the Ukraine noted, *"Sometimes the original developers will just give up, and if there's no way to just continue – for example no one has access to that repository – then you can just fork the project and start a new [one] with a new set of developers."*

Experimentation

This category represents forking as the ability to try new things; either with the intention of merging successful changes back with the original, or simply as a personal exercise. A clear benefit of experimenting on a personal fork is that one cannot cause any unintentional harm to the original project. One developer noted that, if one considers what is often called branches to be forks, then *"As a part of the workflow [a fork] is a vital part. That you can work in peace on your own branch of the program until your feature is ready to be shown to the others."*

A fork may also be necessary in case the project leaders do not share one's views regarding the feasibility of the proposed changes. An experimental fork can then be used to create a proof of concept. As an Italian developer noted:

"If you really don't like something, or if you think that something else has to be done, you always have the chance to try it yourself, even if you are alone [...] and everyone else is against you – why not? You can always try it out and then if it happens that you were right, you can always go back in. So I think it gives a lot of freedom to try out something."

Customization (including forking as a "safety valve")

One central benefit is the possibility to customize a piece of software to better suit one's needs, i.e., to take a project in a new direction. As one developer noted, without the right to fork, *"whoever started the project and defined its goals gets to specify how and when it is used. And in my opinion that is kind of a bit against the whole idea of open source."* Different motivations for customizing a project were noted, among them personal and corporate needs, and disagreements among developers – the kinds of situations for which the right to fork has been called a "safety valve" [9].

[5] Indeed, studies by both Nyman and Mikkonen [15], and Robles and González-Barahona [20] found several cases of such forks.

Personal and corporate needs were of a "scratch an itch" –type. Something close to what was needed already existed, but some modifications were required for it to meet the needs of the developer, or the company they were working for. In discussing such cases, many programmers noted that they commonly involved small tweaks or additions to existing programs. However, a large-scale, continuing development effort based on an existing program was also a possibility, though commonly of a kind covered in the third subcategory, discussed next.

The ability to use the right to fork as safety valve was noted to be beneficial in several different circumstances. One possibility is that a project, while not abandoned, may have stagnated significantly enough that a fork can be perceived as beneficial. As one developer noted:

"We have noticed several times that programs – when they reach a certain age… there comes a certain inertia, and forking may have offered the possibility for a new start. You can leave out the old ballasts, and in that sense many projects that have started as forks may in the end have turned out to work much better [than the original]."

Furthermore, even with actively managed projects, a fork can be beneficial. One example that was frequently brought up was if there was significant disagreement among developers. Cases in which developers disagree on the project's direction, where the difference in question was one of technical aspects relating to the program, were commonly seen as an instance in which it is positive that one can fork, with many explicitly noting that a fork in such a situation may be an entirely favourable thing.

"If the community gets into arguments, or disagree on what direction to take the project, then I think it's fine that they go in different directions [i.e. fork the program]. It might be in the software's best interest; that more is accomplished than [if] they just focus on arguing."

In addition to internal disputes that could lead to, or even necessitate, a fork, programmers also noted its relevance in cases in which an original developer is perceived to mismanage a project. Two example quotes follow.

"First and foremost, [the right to fork] reduces tyranny. If I have created a program and am king and ruler there, then the fact that everyone can fork it means there's a limit to what I can decide about. [...] A second aspect is that it gives power to those who didn't start the project, which is almost the same as the first [point] but slightly different. It means you aren't bound by what someone else wants, which is a pretty central concept in open source [and] free software."

"I think there are a lot of good examples where a forge has been forked and the forked version has just become a lot better than the original. Like… if a project is maintained by somebody that is very single-minded and doesn't really take into account the opinions of others, but the code that they have produced is, like, of high value, I think a fork could be a very good asset there."

5.2. What are the Drawbacks of the Right to Fork?

Common themes brought up were the negative effects a fork could have on users, developers, and the project itself; commonly, the reasoning behind all three groups of answers was related to problems of fragmentation.

Confusion among users

Developers reported that one drawback to forking was the confusion and extra work required to keep abreast of new forks that users experience. In this way, the programmers were both discussing external users and themselves as users of a program.

"You create problems for the users regarding what they should use. For users, a fork is never a good thing. Unless it is done to keep the project alive."

Similarly, even for developers, who one could assume are more informed about software in general, multiple forks means extra effort when researching a project of potential interest. One developer described it thus:

"Well the main thing is of course this fragmentation, it's kind of obvious. I mean if you look at GitHub, some of the projects, you start searching for some project and then you find ten instances of that project and then you would like [wonder], which one do you pick? Then you have to study all ten and see which one is well maintained. Sometimes you want to pick the original because it might have a bigger following. Sometimes the original might have been abandoned, so that would be bad choice, so you actually should pick one of the more recent forks. And it could be that sometimes it was forked precisely because the original was abandoned and no one could contribute to the original project."

Duplicated work among developers

Developers also noted as a drawback the dividing of development efforts resulting from multiple versions of a similar project developed in tandem; something many noted to be the most significant of the drawbacks. As one programmer noted, *"I think the only really big drawback from forking is that then people are contributing to different projects."*

Similarly, discussing situations in which the fork occurs from within a project, one developer noted the dividing of resources to be a drawback, and added that a further *"perhaps even more important"* drawback was arguments between the two groups.

"You can have several years of arguments between the projects, and that can lead to the developers becoming unmotivated, and the users giving up, and the press writing unpleasant things, and things like that. That doesn't usually lead to anything particularly pleasant anymore, and everyone suffers."

Compatibility concerns

A further point brought up was the impact of a fork regarding compatibility. Even in cases where compatibility and co-operation are strived for, over time it was said to be hard to accomplish, with incompatibility, and the resulting duplication of efforts, an unfortunate result. Furthermore, the challenge for third-party software was brought up: cases where a programmer, or company, wants to maintain versions of their program that are compatible with both the original and the fork. The further the forks stray from one another, the more time and resources would be required to remain compatible with both.

Threatening to fork as a negotiation tactic

As forking may potentially have a strong negative impact on a development project (e.g. through a loss of users and duplication of development work), it was noted that the threat of forking can be used as a negotiation tactic:

"Negative part [of forking] is that it's a strain, in a way, to the project. And in some cases it can also be used as a weapon: 'If you don't do what I want…'. And that's the point, especially in companies. The company can always say that 'Yeah, it's open source – do whatever you want. [But] if you don't do what I want, I will fork and take all my force with me'."

Financial challenge

Few developers brought up financial challenges due to the right to fork. This is perhaps not surprising, as the challenge of developing software that is freely available is found in all open source projects, and not exclusively related to forking. As one developer described this challenge: *"The drawback is that it is hard to generate an income in a situation where you can take a copy. I try to sell you an apple; you take a copy of it and eat [the copy]. Well… then you have to kind of think: where does the income come from, then?"*

5.3. When is it Acceptable to Fork?

One of the developers was of the opinion that it was acceptable to fork *"if the original developers weren't doing their job"*, offering by way of example a case in which an open source and proprietary program were being developed in tandem, and then the developers had ceased development of the open source version, focusing their efforts entirely on the proprietary version instead. The programmer commented on such a fork, noting: *"I think it's entirely OK to do that. It keeps the software as free software."*

All of the other programmers interviewed were of the opinion that it is always acceptable to fork. This was surprising considering that they were asked not when it was allowed, or legal, to fork, but rather when it was *acceptable t*o do so. While they could all give examples of problematic aspects related to forking, as well as examples of forks that they felt were unnecessary or would result in problems for the parties involved, they all nonetheless noted that a fork was always acceptable. One Italian developer put it well, *"it's always acceptable. Whether I like it or not."* Another developer raised the question: *"Why would you make it open source if you were against forking?"*

In order to clarify their answers, follow-up questions were asked regarding the kinds of situations under which the developers would consider forking a program they were involved in, and what kinds of forks they would hope could be avoided. This was done to better understand their opinions on what kinds of motivations they considered valid, or even positive, reasons for forking. A basic notion was that there should be a clear point, or reason, to the fork.

"If there is a clear point why people should be contributing to multiple projects then I think it's the only valid reason for forking a project. If there's no real reason why it should diverge from the original, I don't think there is really much point or benefit to be had from forking."

While the developers shared a number of clear justifications, the common thread among them was that they were all situations in which the developers felt that a fork would lead to a better program. As one developer put it: *"I always see the end user as the point, so… If you see that [by forking] we have the chance to do something good, you should."*

Another common notion was that any fork that is significantly different from the original is a sensible motivation.

"If you believe that the new version will get both users and developers, then of course it's sensible [to fork]. [...] The idea is, the point of the fork is, that through it you are actually doing something new, and that it has some meaningful difference."

A situation in which a developer is hindered from making changes they consider meaningful due to divergent views of project goals, or an unresponsive leader, were also commonly brought up. Example quotes from three separate developers follow.

"If the project is for example big – they have a lot of users – they have their own priorities in terms of what to change there. So if you send in your changes there and they wouldn't accept these changes, then you just have to have your own kind of version; your own fork of the project, where you have your own changes."

"Well, if the program doesn't solve the original problem and you can't convince the group working on it that it should be taken in that direction, then you don't have any other choice than to either fork or to choose another program. And in such a situation a fork could be useful."

"If there's a significant difference, or differences in views on in which direction the project should be going, I think that's a good reason why one should be forking a project. Because if the project maintainer doesn't take into account a majority of the projects or the users' of the project's views on in which direction the project should be going, I think it's a good case where somebody should fork it and start directing the project in a way where others could be benefiting more from it."

As one Finnish developer sought to highlight, forking is always acceptable but not always beneficial or practical.

"In the short run forking might be easy – you accomplish a lot in a short time. But in the long run, you lose all the benefits of participating in the activities of a larger community. So in that sense it isn't so much a moral as a practical question."

5.4. When is it Not Acceptable to Fork?

Naturally, the answers to this question closely mirrored those of the previous question. All but one of the programmers noted that it is always acceptable to fork. However, as one Finnish programmer noted, *"That one has the right to fork doesn't exclude there from being situations in which a fork would be problematic"*. Through follow-up questions I attempted to determine what makes a fork undesirable or problematic.

Again, the views closely tracked those in the previous question. Where any fork that is in the best interest of the program was considered a valid motivation, any fork that was not in the best interest of the code was viewed unfavourably. The examples given can be broadly categorized under three headings: in breach of an agreement, personal differences, and malicious forks.

In breach of an agreement

An example of a type of fork one programmer noted to be unfavourably viewed had to do with a situation in which a group of actors would have agreed not to fork. The programmer noted that it would *"perhaps not be that acceptable [...] if one had – through either cooperation, common standardization, or some other common endeavour – decided that some specific open source program should be governed by a non-profit organization, in the interest of keeping it neutral."*

A somewhat related answer by another programmer was regarding not when it was acceptable to fork, but rather what kind of fork a license allows, and that one shouldn't breach license stipulations.

"What isn't ethical is if the licensor decides that this [software] must be kept free, [and] it's modifications must be kept free, and then you fork it and make it closed."

Personal differences

Most developers stated that they viewed forks that were motivated by personal conflict unfavourably. Two examples from the interviews follow.

"In my opinion forks that are based on personal dynamics aren't really that sensible; they kind of weaken the community."

"Many times [a fork is] about people colliding and not features colliding, and that shouldn't be, but... we are human."

Malicious forks

A number of developers brought up the theoretical problem of starting a fork with no other agenda but to compete with the original. That is to say, a fork that did not seek to modify or improve the program in any way, but only sought to create competition. However, most developers also noted that they couldn't think of any examples of when such a fork had actually happened.

"Well of course it's bad if, like, the original project would be good and then someone just goes and forks it to compete with it. But, to be honest, I can't recall seeing that sort of thing happening, where it wouldn't have been justified to fork and, like, diverge from the master. So I mean if that would actually happen I think that would be a bad case but as I personally haven't seen a case like that I don't know if it even exists. I mean do people fork just to compete if the original is a good project? I don't know..."

"Maybe you could come up with some nasty reasons, like someone wants to sabotage the project and just creates extra forks for confusion, or just out of spite, and just draw away some parts of the community towards another project, but... I don't know if that happens. [...] It basically depends on the purpose: if the purpose is OK then forking is OK."

Another example quote emphasizes the imperative among developers to state that forks, while potentially problematic, are always allowed:

"If there is some form of ill will [towards the original] at the base of the fork then I would say it starts being problematic. But even then I think that it, from a legal perspective, is clear that they have every right to [fork] it."

One developer discussed such malicious forks as "sabotage attempts", offering by way of example that he had seen a few instances of single individuals forking a popular operating system and attempting to start an argument among the community which would cause the developers to spend their time arguing rather than coding. However, he noted that such attempts *"maybe succeed for a week or so, but then [the developers] go back to doing what they do."* The developer also discussed what he considered a more serious form of sabotage fork, one that would most likely be linked to financial interests and set in motion by a commercial company with a competing product.

"The first thing is that you achieve a situation in which everyone knows that there is a fork, and you stir up an argument so that

everyone knows that there is something… some problem with the original project, even if there isn't. This is a tactic I expect some large company at some point will use."

One developer's answer offered insight into why so few forks are viewed in a negative light despite the potential downsides to forking: *"Most of the forks – from the outside, it looks like: 'Why? Why don't they all join together and be friends?' From the outside. But when you read up about the reasons and the discussion on the forums, it becomes clear. […] I can't remember any example when I really looked into [a fork] and it would be just… completely senseless."*

5.5. Are There any Moral Obligations When Forking?

Contrary to expected, in all cases but one these interviews did not yield explicit lists of issues programmers considered significant to either remember or consider when forking. In fact, one developer began his answer by noting that he needed some time to consider the question, as he had never before given it any thought. In light of the insights gained from previous answers, particularly that of programmers' strong feelings of the right to fork as integral to open source, it is perhaps not surprising that no such unwritten yet widely adopted "moral obligations rulebook" exists[6].

However, there was one issue that all hackers brought up: crediting the original contributors. This seemed a self-evident point. Indeed, the aforementioned programmer who noted that he hadn't pondered what moral obligations one may have, noted that one should credit previous contributors, but added that the point was "*kind of obvious*." Some connected the obligation to give credit with the stipulations of the license, others with a moral obligation to do so. Example answers included:

"Just what the license says. They need to credit the author if the license requires it but I think that's it."

"What is important is to give credit to the original project, and meticulously document what has come from whom – who has contributed these different parts. To give credit where credit is due."

As mentioned previously, there was one notable exception. This was from a developer who began his answer by noting that the answer *"depends entirely on how one views the freedom of the program. I am what you would call a free software person rather than an open source person."* The programmer then went on to note that:

"From a free software perspective, one has the moral responsibility to see to it that those who use the program aren't hurt. So for instance that one maintains the fork as free software, and sees to it that, if possible, one keeps both versions compatible with one another […] so that, if possible, one cooperates with the original project. […] If one can do these kinds of things, then I think one has the moral imperative to do them. But the important [thing] is to make sure that the users – not the developers, the users – don't suffer."

5.6. Is There a Proper Code of Conduct if One Wants to Fork a Program?

Only one of the developers, the one who noted he was a free software person rather than an open source person, noted a number of particular actions that he felt should be taken regarding the actual process of starting a fork. He began by stating: *"One should be honest and clear about why one is doing a fork. That means that the big Internet – they won't have to guess, and find scandals where they don't exist."* He continued by noting that the fork should be done *"in such a way that the original does not suffer unnecessarily."* This included practical issues aimed at avoiding confusion and problems among developers and users, with examples being setting up a new mailing list, website, server to host the code, and choosing a new name for the project. *"By and large,"* he concluded, *"one should behave like a human being."*

While other hackers didn't offer views regarding a code of conduct when forking, something that was brought up by several programmers was what to do *before* one decides to fork: to ensure that a fork is absolutely necessary. This point was brought up by both developers who had forked programs as well as those who had not. Examples from the interviews follow.

"Well, usually you start by seeing if you can avoid the fork: talk to the developers and say 'we would like to do this'. And then, if you can't agree, then you say 'OK, we'll fork'."

"My feeling would be that, if the project is active and has a community, it makes sense to talk to them to see if your version would really be different. Because if you don't do that, then it might… there are these drawbacks – that it creates this, kind of, incompatibility and, it kind of makes the whole landscape more complex. So you have to make sure that it's worth it. So it makes sense to talk to the original developers and see that, 'Can't we cooperate?'"

A further point that was brought up by several programmers was the importance of maintaining some form of communication with the original so that one can, as one interviewee put it, *"Share the parts of the program that can be shared"*. Indeed, one programmer, in discussing a fork that had been made from a project that he had spent the better part of a decade working on, noted that, for many years they were able to share code. In discussing the fork, he only bemoaned a decision made by the fork project's leader many years after originally forking, to resolve a technical issue in such a way that they would no longer be able to share code with one another.

Regarding communication before a fork, one developer offered a significantly different viewpoint, light-heartedly noting that *"I would personally dislike it if every person that would want to fork some project [of mine] would be contacting me. Just a lot of spam. I would have to set up some email account that would auto-reply 'yes' and… there wouldn't be much value to be had from that"*.

While this position may seem to contradict earlier points, it seems likely that the approach programmers' would consider favourable could be relative to the proximity of the goals of the original and the fork: the less planned overlap there is between the two, the less important it is to initiate and maintain significant contact with one another.

[6] There are, however, indications that even among those without an explicit list, an implicit list may still exist. Further research would be needed to confirm this, but it could include themes like: thou shalt not infringe on the right to fork; agreements shall be binding; personal differences shall be moot; and, don't start a fork out of spite.

6. CONCLUSIONS: SUMMARY OF RESULTS

What is good/bad about the right to fork?

First and foremost, developers consider the right to fork to be integral to the very concept of open source software. Beyond the general view of the right to fork as an important open source cornerstone, the right to fork is considered positive for several reasons. Among them are the possibilities it offers to: preserve abandoned programs; to experiment with and customize existing programs; and to both minimize tyranny and resolve disputes regarding development efforts through giving the involved parties the opportunity to develop their own versions of the contested program.

While considered vital, the right to fork can still be problematic. Examples of related concerns are confusion among users, and extra work among developers, including both duplication of efforts and increased work if attempting to maintain compatibility. Furthermore, particularly if the fork is from within the project, the resulting decrease in developers is a concern. The right to fork also has a significant effect on the ability to generate income for a project.

What kinds of forks are viewed favourably/unfavourably?

The kinds of forks viewed favourably are those where there is a reason to believe the fork will lead to a better program. Examples given of such scenarios included: that the program better serves (all or some) users, that the original is no longer being developed or development has stagnated, and that disagreements among developers significantly hamper or halt its development.

The kinds of forks that hackers would prefer to be avoided are forks that are based on personal issues (e.g. rather than differences of opinion regarding which direction to take a development in), and forks that are not so significantly different that they couldn't have remained one single program. Purely malicious forks, created merely to sabotage or confuse, are also frowned upon as a concept, but such forks seem exceedingly rare.

How should one go about forking? Are there any moral obligations or is there an (unwritten) code of conduct?

The sole rule that seems to guide forking overall is that one must give credit to those that have worked on the program previously (something also stipulated by license). However, the value of being in contact with the original developers before one decides to fork, to see if the fork is in fact a necessity, was underlined by several developers. This may lead to avoiding the fork and thus keeping more developers working on the same project. In cases in which forking is nonetheless necessary, the projects may still be able to benefit from one another's efforts by sharing code.

7. DISCUSSION, LIMITATIONS, AND FUTURE WORK

While not all programmers gave answers that were identical in content, no programmer's answers were counter to views expressed by the others. No kinds of forks were viewed as positive by some yet negative by others; no issues about forking were viewed as positive by some and negative by others. Thus, there was a general consensus among all of the programmers interviewed.

The findings of this study largely support what can be found in practitioner literature and earlier studies. Gamalielsson and Lundell's [3, 4] findings, that a fork can be sustainable over time and that an open source software community can outlive a project, fit seamlessly with the interview answers gathered for this paper. Particularly in the event of a program's governing body being perceived as acting against the interests of the community at large, a fork not only seems likely, but it may generate more wide-spread support among developers than is enjoyed by the original development.

The data also offers further support for the findings of this author's earlier paper [12], that a fork is considered valid in cases where there is the possibility of a decrease in openness of a program, either through changes in licensing or through the community's hampered ability to contribute. The views expressed by the interviewees also corroborate earlier conclusions drawn by this author and Mikkonen [15], that programmers principally fork code in order to achieve pragmatic, development-related goals, rather than in order to compete with existing communities.

The interviewees' permitting stance overall towards any fork may seem to contradict existing literature about the social pressure to avoid forking. However, this discrepancy seems likely to find its explanation in the broad scope of extant interpretations of the term fork. In popular use the term has grown to encompass a greater breadth of actions than only competitive forks, yet existing literature commonly uses a much more narrow interpretation, that of an internal split among developers due to irreconcilable differences [13]. Thus, these findings do not so much contradict earlier statements regarding programmers' avoidance of forking as they do highlight the evolving interpretation of the meaning of the word fork. Illustrating this point, one of the developers interviewed, having read an early version of this paper, noted that on GitHub the number of times a project has been forked is used as one measure to rank its popularity. Furthermore, he noted that developers actively encourage others to fork their programs.

Very early on in conducting the interviews it became clear that more interviews would likely garner new ways of expressing similar points rather than new viewpoints. While I initially suspected that this similarity among answers may have something to do with a common "hacker ethic", based on the data gathered, the imperative to avoid a fork seems to be less an extrinsic community pressure and more an intrinsic sense of pragmatism shared by all developers.

A final interesting point to note is that the more strict views with regards to what is acceptable and not acceptable in regards to forking were offered by a developer with an affinity for free software, rather than open source software. Though the differences in views were not great, where they did exist the main emphasis of those differences, perhaps not surprisingly, was on the maintained freedom of the code.

7.1. Limitations and Future Work

There were no interview questions that focused specifically on what developers think about variations in open source licenses. Thus, no data was gathered specifically on forks from open source projects that are later made proprietary. Arguably, if developers had strong opinions about such forks, they would have voiced them even given the questions not specifically noting them, as was the case with one of the interviewees. Furthermore, several developers discussed the benefits for companies in that they can fork and modify existing programs. Nonetheless, future work could more specifically address developers' views on proprietary forks of open source software programs.

A further limitation of the study relates to the pool of interviewees and its limit in scope. It would be interesting to include more developers representing a wider distribution of nationalities. However, given the homogenous nature of the interviewees' answers despite them coming from four countries, it is uncertain that a modest increase in interview subjects or backgrounds would have had a significant effect on the outcome. Future work could broaden the sample, including a wider mix of countries and backgrounds. The similarity of the opinions among interviewees in this study makes it challenging to suggest where differences of opinion might exist, or be particularly pronounced; which is something that further study could also reveal. However, the data does seem to indicate that a deeper analysis of the potential differences of opinion between free software versus open source software enthusiasts might offer at least one further interesting avenue of research.

The current study does not attempt to classify opinions relative to their importance. A quantitative, survey-based analysis, based on the information gained here, could uncover information about such issues. Such a study could also shed light on what variables may correlate with certain opinions: do programmers of a certain age, background, or work-history have more similar views than those from other groups? What about programmers who consider themselves more aligned with either the term free software or open source software? While the data from this article suggests that identifying significant differences of opinion is not a certainty, it would be interesting if such differences were uncovered.

8. REFERENCES

1. Fogel, K. 2006. *Producing Open Source Software*. O'Reilly, Sebastopol, CA.
2. Fung, K.H., Aybuke, A., and Tang, D. 2012. Social Forking in Open Source Software: An Empirical Study. In *Proceedings of the CAiSE'12 Forum at the 24th International Conference on Advanced Information Systems Engineering (CAiSE)* (Gdańsk, Poland, June 28, 2012).
3. Gamalielsson, J. and Lundell, B. 2012. Long-Term Sustainability of Open Source Software Communities beyond a Fork: A Case Study of LibreOffice. In: *Proceedings of the 8th IFIP WG 2.13 International Conference*, *OSS 2012* (Hammamet, Tunisia, 2012).
4. Gamalielsson, J. and Lundell, B. 2014. 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
5. Himanen, P. 2001. *The hacker ethic, and the spirit of the information age.* Random House, New York, NY.
6. Lerner, J. and Tirole, J. 2002. Some Simple Economics of Open Source. *The Journal of Industrial Economics*, Vol. 50, No. 2, pp. 197-234.
7. Levy, S. 1974. *Hackers*. O'Reilly, Sebastopol, CA.
8. Meeker, H. 2008. *The Open Source Alternative*. John Wiley & Sons, Inc. Hoboken, NJ.
9. Moen, R. 1999. *Fear of Forking essay*. Available at: http://linuxmafia.com/faq/Licensing_and_Law/forking.html
10. Moody, G. 2009. Who Owns Commercial Open Source – and Can Forks Work? *Linux Journal*, 23 April, 2009.
11. Moody, G. 2011. The Deeper Significance of LibreOffice 3.3. *Computerworld UK*, 28 January, 2011.
12. Nyman, L. 2013. Freedom and forking in open source software. *Proceedings of the Nordic Academy of Management Conference* (Reykjavik, Iceland, 2013).
13. Nyman, L. 2014. When is a Fork a Fork? On the Definition(s) of Code Forking. Forthcoming.
14. Nyman, L. and Lindman, J. 2013. Code Forking, Governance, and Sustainability in Open Source Software. *Technology Innovation Management Review*, January, pp. 7-12.
15. Nyman, L. and Mikkonen, T. 2011. To Fork or Not to Fork: Fork Motivations in Sourceforge Projects. In: Hissam, S. (Ed.), IFIP Advances in Information and Communication Technology, vol. 365. Springer, Heidelberg. pp. 259-268.
16. Nyman, L., Mikkonen, T., Lindman, J., and Fougère, M. 2011. Forking: the invisible hand of sustainability. In: *Proceedings of the OSS 2011: Towards Sustainable Open Source, Sustainability workshop* (Salvador, Brazil, 5 October, 2011).
17. Nyman, L., Mikkonen, T., Lindman, J., and Fougère, M. 2012 Perspectives on Code Forking and Sustainability in Open Source Software. In: Hammouda, I. (Ed.), IFIP Advances in Information and Communication Technology, vol. 378. Springer, Heidelberg. pp. 274-279.
18. Raymond, E. 1999. *The Cathedral and the Bazaar*. O'Reilly Media, Inc., Sebastopol, CA.
19. Raymond, E. 2001. *How to Become a Hacker*. Available at: http://www.catb.org/esr/faqs/hacker-howto.html
20. Robles, G. and González-Barahona, J.M. 2012. A Comprehensive Study of Software Forks: Dates, Reasons, and Outcomes. In: Hammouda, I. (Ed.), IFIP Advances in Information and Communication Technology, vol. 378. Springer, Heidelberg. pp. 1-14.
21. St. Lauren, A. 2004. *Open Source & Free Software Licensing*. O'Reilly, Sebastopol, CA.
22. Stewart, K.J. and Gosain, S. 2006. The Impact of Ideology on Effectiveness in Open Source Software Development Teams. *MIS Quarterly,* Vol. 30 No. 2, pp. 291-314. June 2006.
23. Weber, S. 2004. *The Success of Open Source*. Harvard University Press, Cambridge, MA.
24. Wheeler, D. 2007. *Why Open Source Software / Free Software (OSS/FS, FLOSS, or FOSS)? Look at the Numbers!* Available at: http://www.dwheeler.com/oss_fs_why.html
25. Ågerfalk, P.J. and Fitzgerald, B. 2008. Outsourcing to an Unknown Workforce: Exploring Opensourcing as a Global Sourcing Strategy. *MIS Quarterly* Vol. 32 No. 2, pp. 385-409/June 2008.

Appendix 1. Interview questions

The interviews were semi-structured, allowing for follow-up questions when needed. Examples of follow-up questions are given as bullet points. The interview began with three control questions, each with possible follow-up questions. The first question was about how they got into programming, followed by a question about their open source background. These questions were both to confirm that the interviewees met the criteria established for interviewees, as well as to establish some further details of their level of experience with programming in general and open source in particular. The third question was about their definition of a fork.

The questions were as follows:

1. When, in your opinion, is it appropriate to use the term "fork"?
   - Does it matter to the definition whether I plan to merge the program back with the original?
   - Does it matter to the definition whether I plan to modify the program?
   - Is it a fork if a functionality has been reused as part of the new program, but it is buried/invisible inside it, or at least not prominent?
     - o What if the functionality is the entire forked program vs. only part of it?
   - Does the term require community involvement? Is it a fork if I take code for a personal program that I never intend to share with anyone else?
2. What are the benefits of the existence of forking as a practice?
3. What are the drawbacks of the existence of forking as a practice?
4. When, or under which circumstances, is it acceptable/OK to fork a program?
   - Have you ever forked a program, or been involved in a forked program?
   - Under what kinds of circumstances would you yourself consider forking a program in which you were participating?
   - Can you think of any examples of forking where you felt it was good that a fork had happened?
5. When is it not acceptable/OK to fork a program?
   - Under what kind of circumstances would you consider it a bad thing that a fork happened?
   - What kinds of forks do you hope could be avoided?
   - Can you think of any instances of forking which you felt it was a bad thing that a fork had happened?
6. When someone forks a program, are there any moral obligations towards anyone? The original developers? The community? Others?
7. Is there a proper code of conduct when one wants to fork a program, and if so, what is it?