To Fork or Not to Fork: Fork Motivations in SourceForge Projects

Linus Nyman, Tommi Mikkonen

To Fork or Not to Fork: Fork Motivations in SourceForge Projects

Linus Nyman[1] and Tommi Mikkonen[2]
1 Hanken School of Economics, Helsinki, Finland
linus.nyman@hanken.fi
2 Tampere University of Technology, Tampere, Finland
tommi.mikkonen@tut.fi

Abstract. A project fork occurs when software developers take a copy of source code from one software package and use it to begin an independent development work that is maintained separately from its origin. Although forking in open source software does not require the permission of the original authors, the new version, nevertheless, competes for the attention of the same developers that have worked on the original version. The motivations developers have for performing forks are many, but in general they have received little attention. In this paper, we present the results of a study of forks performed in SourceForge (http://sourceforge.net/) and list the developers' motivations for their actions. The main motivation, seen in close to half of the cases of forking, was content modification; either adding content to the original program or focusing the content to the needs of a specific segment of users. In a quarter of the cases the motivation was technical modification; either porting the program to new hardware or software, or improving the original.

1 Introduction

A project fork takes place when software developers take a copy of the source code from one software package and use it to begin an independent development work. In general, forking results in an independent version of the system that is maintained separately from its origin. The beauty of open source software development is that no permission from the original authors is needed to start a fork. Therefore, if some developers are unhappy with the fashion in which the project is being managed, they can start an independent project of their own. However, since other developers must then decide which version of the project to support, forking may dilute the community as the average number of developers per system under development decreases.

Despite some high-visibility forks, such as the forking of OpenOffice (http://www.openoffice.org/) into LibreOffice (http://www.libreoffice.org/), the whole concept of forking has seen little study. Furthermore, developers' motivations for forking are understood even less, although at times it seems rational and straightforward to identify frustration with the fashion in which the main project is being managed as a core reason.

In this paper, we present the results of our investigation of SourceForge (http://sourceforge.net/) for forked projects and the motivations the authors have identified for performing a fork. Furthermore, we categorize the different motivations and identify some common misbeliefs regarding forking in general.

The rest of this paper is structured as follows: Section 2 discusses the necessary background for explaining some of the technical aspects associated with forking, Section 3 introduces the fashion in which the research was carried out, Section 4 offers insight into our most important findings, and Section 5 discusses them in more detail. Section 6 proposes some directions for future research, and Section 7 concludes the paper with some final remarks.

2 Background

When pushed to the extreme, forks can be considered an expression of the freedom made available through free and open source software. A commonly associated downside is that forking creates the need for duplicated development efforts. In addition, it can confuse users about which forked package to use. In other words, developers have the option to collaborate and pool resources with free and open source software, but this is enforced not by free software licenses, but only by the commitment of all parties to cooperate.

There are various ways to approach forking and its study. One is to categorize the different types to differentiate between, on the one hand, forks carried out due to amicable but irreconcilable disagreements and interpersonal conflicts about the direction of the project, and on the other, forks due to both technical disagreements and interpersonal conflicts [1]. Still, the most obvious form of forking occurs when, due to a disagreement among developers, a program splits into two versions with the original code serving as the basis for the new version of the program.

Raymond [2] considers the actions of the developer community as well as the compatibility of new code to be a central issue in differentiating code forking from code fragmentation. Different distributions of a program are considered ‘pseudo-forks’, because at first glance they appear to be forks, but in fact are not, since they can benefit enough from each others’ development efforts not to be a waste, either technically or sociologically. Moody [3] reflects Raymond’s sentiments, pointing out that code fragmentation does not traditionally lead to a split in the community and is thus considered less of a concern than a fork of the same program would be. These sentiments both echo a distinction made by Fogel [1]: it is not the existence of a fork which hurts a project, but rather the loss of developers and users. Here it is worth noting, however, that forking can potentially also increase the developer community. In cases in which developers are not interested in working on the original (for instance due to frustration with the project direction, disagreements with a lead developer, or not wanting to work on a corporate sponsored project), not forking would lead to fewer developers as the developers in question would likely simply quit the project rather than continue work on the original.

Both Weber [4] and Fogel [1] discuss the concept of forks as being healthy for the ecosystem in a 'survival of the fittest' sense; the best code will survive. However, they also note that while a fork may benefit the ecosystem, it is likely to harm the individual project.

Another dimension to forking lies in the intention of the fork. Again, several alternatives may exist. For instance, the goal of forking can be to create different branches for stable and development versions of the same system, in which case forking is commonly considered to serve the interests of the community. At the other extreme lies the hostile takeover, which means that a commercial vendor attempts to privatize the source code [5]. Perhaps somewhat paradoxically, however, the potential to fork any open source code also ensures the possibility of survival for any project. As Moody [6] points out, the open source community and open source companies differ substantially in that companies can be bought and sold, but the community cannot. If the community disapproves of the actions of an open source company, whether due to attempts to privatize the source code or for other reasons related to an open source program, the open source community can simply fork the software from the last open version and continue working in whichever direction it chooses.

3 Research Approach

In the study, we used SourceForge (http://sourceforge.net/) as the repository of open source programs from which we collected forks. SourceForge contains over 260,000 open source projects created by over 2.7 million developers. Creating new projects, participating in those that already exist, or downloading their contents is free, and developers exercise this freedom: programs are downloaded from SourceForge at a pace of more than 2,000,000 downloads daily.[1]

SourceForge offers programmers the opportunity to briefly describe their program, and these descriptions can be searched using keywords. Using this search function, we compiled a list of all of the programs with the word "fork" – as well as dozens of intentionally misspelled variations of the word fork, none of which turned up any hits – in their description. We then analyzed all the descriptions individually to differentiate between them and to sort out programs that the developers claimed had forked their code base from another program (which we call "self-proclaimed forks") from those which included the term 'fork' for some other reason, either to describe a specific functionality of the program or as part of its name (i.e. false positives). Consequently, a program that stated "This is a fork of …" was considered a fork, while a program which noted that it "…can be used to avoid common security problems when a process forks or is forked" was not. If it was impossible to categorize a project based on the available data, it was discarded. Our data consisted

[1] Source: http://sourceforge.net/about, accessed March 9, 2011

of all programs registered on SourceForge from its founding in late 1999 through 31 December 2010, resulting in a time span of slightly more than 11 years. This search yielded a total of 566 programs that developers report to be forked.

We then analyzed the motivations stated in the descriptions of the forked programs. The coding process was done in three phases. First, we went through all of the descriptions and wrote a brief summary of the motivations, condensing the stated reasons to as few words as possible. Then, we went through all of the motivations and identified common themes, or subgroups of motivations, among them. In cases where the fork included elements from more than one theme, we placed it in the subgroup that seemed the most central to the motivation behind the fork. Finally, we examined the subgroups to identify overarching groups of themes.

To give some examples of the coding, one fork stated: "[Project name] is a fork of the [original project name] project. [The] purpose of [project name] is to add many new features like globule reproduction, text to speech, and much more." The motivation behind the fork was identified as belonging to the subgroup "add content", which in the final step was combined (with a subgroup of programs which sought to focus content) into a group called content modifications. A fork which sought to fix bugs, and a fork which was motivated by porting a program, were first put into separate subgroups, "technical: improvement" and "technical: porting", and then these subgroups were combined into the "technical modifications" group. Further examples from the data are presented in the next section.

Based on the descriptions entered by the developer, we were able to identify motivations for 381 of the forks. The group of forks which we were unable to categorize consisted of two main types of descriptions: firstly, descriptions which offered no insights as to underlying motivations, e.g. programs which simply stated which program they were forked from; secondly, cases in which it was unclear from the description if the elements described were added in the fork or if they existed in the original; in other words, one couldn't determine if the description of the program included the motivation behind the fork, for instance new technical features, or if they were describing pre-existing features common to both the original and the fork.

4 Reasons for Forking

Based on the data obtained, developers commonly attribute their reasons for forking the code to pragmatism. For a variety of reasons, some of which were well documented and some of which were unclear, the original version of the code failed to meet developers' needs. To expand the scope of the system, the developers then decided to fork the program to a version which serves their own needs. The descriptions of the forks include programs which note that certain changes have been made to the fork, as well as those programs which discuss which changes will or should be made to the forked version. In this paper, we have not distinguished between the two: both planned and already implemented changes are treated equally, since the goal was to study motivations rather than eventual implementations. In

general, the forks appear to stem from new developers rather than the original developing team splitting into two camps. In fact, the data contain almost no references to disagreements among developers that might have led to the fork. However, this does not mean that such disagreements could not have existed.

In the following section, we provide a more detailed view of the different motivations we were able to find in the data (n = 381). The main motivations fall into two large groups (content and technical modifications) which comprise nearly three quarters (72%) of all forking motivations. Four smaller groups, all of similar size, comprise an additional 23% of the motivations. These four groups included the reviving of a project, license- or FOS-related motivations, language- or country-related reasons, and experimental forks. The remaining motivations, grouped simply as "other", consisted of diverse yet uncommon reasons. An overview reflecting the numbers of forks appears in Figure 1.

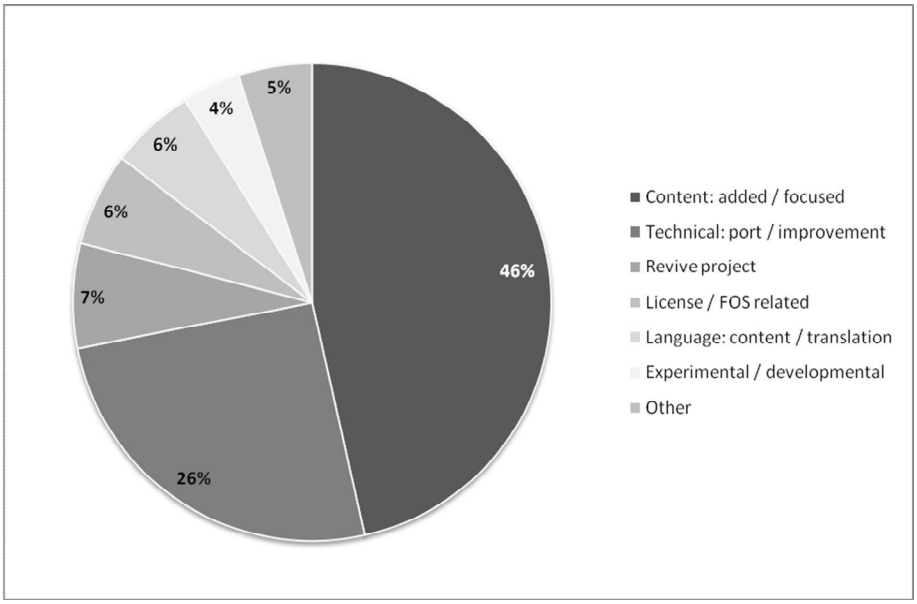

Fig. 1. Fork motivations in SourceForge projects

4.1 Content modifications

Comprising almost half of all forks, content modifications is the largest group. The two main subgroups within the content modifications category, both of which are nearly equal in size, were the adding and the focusing of content; these are briefly discussed below.

Adding content is a self-explanatory reason for making a fork. The developers added new features or other content (e.g. adding better documentation, helper utilities, or larger maps to a game). Quite often, developers didn't describe additions in detail; one developer, for instance, simply noted that the program was a fork "that has the features I'm missing from [the original]." Another developer stated that the fork was "A [program name] fork with more features". In several cases, this group of forks also included bugfixes.

Focusing content implies focusing on the needs of a specific user segment. This category includes forks with both a technical and content-related focus, along with the addition of functionalities and features as well as the removal of elements or features unnecessary for a specific segment or purpose. Examples of content-related focus include programs forked in order to focus on serving the needs of dance studios, radio talk shows, catering companies, program developers, and astronomers, to name but a few. Examples of technical focus include forks "aimed at higher-resolution iOS devices", a fork which "features improvements and changes that make it more oriented for use in a Plone intranet site", and a fork intended "to run on machines that have 800x600 screen resolution". In a minority of the cases in the focusing content category, the original program was forked mainly to remove elements from the original. The main goal in this group was to create a lighter or simpler version of the original, with speed and ease of use as the main focus. One developer stated that the fork was "lightweight, less bloated" and that it was forked to "make [the original] simpler, faster, more useable." Another developer noted that the fork was "Smaller, faster, easy to use."

4.2 Technical modifications

This group, comprising just over a quarter of all forks, can be divided into two subcategories: porting and improving. A characteristic of this category was that little if anything was visibly different to the user; the forked programs simply focused on either porting or improving the original.

Porting the original code to new hardware or software was the more common of the technical motivations for forking, usually involving porting the original to fit a certain operating system, hardware, game, plug-in, migrate to a different protocol, or other such reasons. Examples from the data for this group include a "fork of [program name] to GNU/Linux", a fork "compatible with the NT architecture", "a simple C library for communicating with the Nintendo Wii Remote […] on a Linux system", and a program fork whose main target was to create a version "which works with ispCP." Some forks were ported to reduce a dependency; for instance, one developer who noted that the fork was "geared towards 'freeing' [the original program] from its system dependence, [thus] enabling it to run natively on e.g. Mac OS X or Cygwin." Another developer noted that the program was forked because the developer could not find a "good and recent [program type] without KDE dependency."

Improving the original program was slightly less common in the technical motivations category than porting, which focuses on improving already existing features and contains mostly bugfixes, code improvement and optimization, and security improvements. Some cases were very general in their descriptions, noting only that it was an "upgraded" or "improved" version of the original, or that the code was forked "to fix numerous problems in the code" or to "improve the quality of emulation". Others were more specific, as with the developer of one fork, who notes that "The main goal is to build a new codebase which handles bandwidth restrictions as well as upcoming security issues and other hassles which showed up [during] the last 6 months."

4.3 Reviving an abandoned project

The third common motivation for forking was to continue development of a project considered abandoned, deceased, stalled, retired, stagnant, inactive, or unmaintained. In several of these cases, the developers of the fork note who the original developers are and credit them. In a few cases, the developers of the fork note that they attempted (unsuccessfully) to reach the original developers; in other words, forking the code was the last available option for these developers, as the original developers could no longer be reached. One such example is a fork which the developer notes was "due to long-time inactivity" and then goes on to state "We want to thank the project founder [name] for starting this project and we intend to continue the work". In another case, also due to the inactivity of the original developer, the developer of the fork acknowledges the original author and notes that the fork "includes changes from comments made on his forum." Other examples from the data are: "This project is a fork of the excellent but dead [project name] project", "This project is a fork of the stalled [project name]", "a code fork from the (deceased) [project name] source", and, finally, "The previous maintainer is unresponsive since 2008 and the library [has] some deficiencies that need to [be] fixed. Anyway, thanks for creating this great library [name of original developer]!"

4.4. License/FOS-related issues

This group consists of forks which were motivated by license-related issues or a concern for the freedom of the code. Some of the forks appear to be simply a form of backup copies: stored open source versions of well-known programs. The motivation for this subgroup was a concern that the original version might become closed source. In one case, the developer stated that the fork was due to concern about the future openness of the code. In a similar case, a developer noted about the fork that "This is a still-GPL version of [program name,] just in case." One fork simply identifies the motivation as a "license problem". In five cases, the program was forked because the original was deemed to have become either closed source or

commercial, and in one case, developers noted that the fork occurred because certain bits of the original code were closed source. One fork notes that the new version removes proprietary (boot) code from the program, but that "there is no need to use this version unless you are concerned about the copyright status of the embedded boot code."

4.5 Language- and/or country-specific modifications

A small group of the forks were motivated by language and country. This group could well be considered a subcategory of the "focusing content" group, but was considered separate due to its clear language-related focus. The simplest, though not most common, form of forks included programs which were merely translated into one or more languages; in most cases, however, new content was also added to customize the fork for a specific country and/or group. Some examples are forks created for elections in New Zealand, the right-to-left reading of Hebrew texts, and a program "customized to meet German requirements regarding accounting and financial reporting."

4.6 Experimentation

This group consisted mostly of forks which declared that they existed for experimental purposes, with a handful citing development reasons. A feature common among many of these forks is that the developers state that the fork is temporary and that successful new features or improvements will be incorporated into the original program. Some describe the fork as simply "for testing", while others go into greater detail, noting for instance that the fork is "aimed at experimenting with a number of features turned up to maximum." One developer notes that the fork is simply "for fun", and then goes on to tell readers where they can find the original project.

4.7 Other reasons

Of the remaining forks, a handful described it as a "community fork." In some of these cases, it was possible to identify an overarching motivation behind the community fork; in others it was not, the implications of the term in those cases remaining unclear. Two cases cite a reprogramming in a different programming language as the reason for the fork. The remaining reasons for the forks defied categorization, and included such motivations as a desire to create a study tool for the developer, as well as to test SourceForge for a different project.

Finally, the most surprising of the remaining groups was the group motivated by disagreement or breach of trust. In the beginning of the study, we assumed that a significant number of forks would stem from disagreement between developers. In reality, we were able to identify such forks, but their proportion is quite small: we

identified only four cases, three of which stated that the users sought something the original developers did not intend to implement and one which noted that the fork was a reaction to a breach of trust. Furthermore, even some of these cases may be attributed to the original developers' loss of interest in the project.

5 Discussion

The data in this paper are based on information provided by developers themselves. Many of the cases of self-proclaimed forking – such as when a developer continues an abandoned project – could arguably be defined as something other than a true fork. However, determining forks any other way (other than through the self-proclaimed approach used here) would require a technical definition of a fork that would have to be mined from the project data. At present, no such mechanism seems to exist, and in general, differentiating between forked and fragmented code is an ambiguous practice, unless defined by elements outside of the code itself. Consequently, we have identified the developers as the most reliable source of information, at least at present.

Beyond the challenge of defining a fork, one here also needs to note two issues: how the choice of SourceForge as a sampling frame might affect the data, as well as how accurate, or complete, the descriptions offered there are. The choice of SourceForge could affect the data in several ways. The main question would seem to be whether the characteristics of the average program – or program fork – on SourceForge differ from those of programs hosted on other sites, or from independently hosted programs. For example, given that larger projects often have their own hosting, it is possible that we are seeing only a small number of forks in some categories because projects that would face such issues are not using SourceForge. As to the completeness of the motivations offered by developers, there could be a number of reasons why the information offered is incomplete. For instance, the low frequency of disagreements as a motivational factor in forking may perhaps in part be explained by either a reluctance to mention such disagreements or the limited space offered by SourceForge in which to describe the program. It is also possible that such information, while not stated on SourceForge, would be available on project homepages. Indeed, we came across a project which noted elsewhere that a disagreement among the developers of the original was a factor in the fork; however, the same project did not mention this disagreement in their description on SourceForge.

In general, the results of our study suggest that forking is not a particularly extreme situation in real-life projects. For the most part, developers' motivations are easily understandable, and forking can be considered a reasonable action. However, this does not mean that hostile takeovers are absent from high-profile projects, but simply that in the vast majority of cases, developers appear simply to seek to satisfy their own needs and to develop interesting systems. Such motivations were evident

in the documentation in many ways. Some of the forks note that the changes or improvements have already been made, whereas others announce the intended direction of the fork and mention features to be added to it. Furthermore, crediting the original developers was a rather common practice among those who forked a program, which further emphasizes the fact that forks sought to achieve certain goals, not to compete with existing communities. Perhaps more telling still is that a number of forks noted that they hoped to be temporary, and clearly stated their desire that the bugfixes and improvements introduced in their fork be incorporated into the original program.

6 Future Work

Future work regarding issues associated with forking could take numerous directions. Below we list some of the most promising directions that merit further investigation.

Defining a fork. All of the programs in the data for this article define themselves as forks. In practice, upon more careful review, many of them could perhaps more accurately be categorized as pseudo-forks, code fragmentation, or simply different distributions of a code. The creation of a commonly agreed-upon view of forking vs. fragmentation (or distributions) vs. code reuse would be a very practical step that could benefit both researchers as well as the entire open source community. It could also be possible to define a fork based on technical details, rather than depend on information provided solely by the developers.

Licenses before and after forking. Future researchers could conduct a survey of developers who have forked a program in which they explain their choice of license in comparison to the license of the original program from which they forked.

Perception of forking. Another practical aspect related to forking is how programmers view it; in other words, when is it acceptable to fork, and when is it not? Furthermore, discovering whether certain behaviors make forking more acceptable among developers would be an important direction for such work.

Expanding the data set. Performing a similar study for other sites that host open source projects would contribute to a deeper understanding of forking. Because all the data come from only one source, certain aspects may skew the results. Furthermore, it would be interesting to test if one can tie the observed categories to antecedents or consequences, e.g., are particular kinds of software more likely to fork in particular ways or are particular kinds of forks more successful?

Forking in relation to business. A number of forks we have identified occurred because the original project became closed source. Examining what happened to these projects would deepen our understanding and view of forking in relation to business.

7 Conclusions

Forking is one of the least understood topics in open source development. While often perceived initially as something malicious, the developers who perform the actual forking cite rather straightforward reasons for their actions.

In this paper, we addressed the motivations of developers for performing a fork. The data used in the project originate from SourceForge (http://sourceforge.net/), one of the best-known hosts of open source projects, and focus on "self-proclaimed forks", or programs that the program developers themselves consider to be forks. The motivations behind forking are based on developer input, not on mining technical qualities of the project. However, using only the latter to determine forking would be difficult, as separating forking from other open source-related phenomena is problematic and inconclusive. At the very least, additional data from developers are needed to define forking.

In conclusion, while hostile takeovers and the hijacking of a project as well as a loss of developers after a fork are often associated with forking, the reality is that forks seem to be a lot less dramatic. In fact, forking appears to be more or less business as usual, and developers fork because doing so provides certain benefits for their own goals. While we were able to find forks where the rationale for forking lay in disagreement or trust issues, such cases were few in comparison to the total number of projects we studied.

References

[1] Fogel (2006) Producing Open Source Software. O'Reilly, Sebastopol, CA.
[2] Raymond (2001) The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. O'Reilly, Sebastopol, CA.
[3] Moody (2011) The Deeper Significance of LibreOffice 3.3. ComputerWorld UK, January 28.
[4] Weber (2004) The Success of Open Source. Harvard University Press, Cambridge, MA.
[5] Lerner and Tirole (2002) Some Simple Economics of Open Source. The Journal of Industrial Economics, Vol. 50, No. 2, pp. 197-234.
[6] Moody (2009) Who owns commercial open source and can forks work? Linux Journal, April 23.