A Comprehensive Study of Software Forks: Dates, Reasons and Outcomes

Gregorio Robles, Jesús M. González-Barahona

A Comprehensive Study of Software Forks: Dates, Reasons and Outcomes

Gregorio Robles and Jesús M. González-Barahona

GSyC/Libresoft, Universidad Rey Juan Carlos
{gregorio.robles,jesus.gonzalez.barahona}@urjc.es

Summary. In general it is assumed that a software product evolves within the authoring company or group of developers that develop the project. However, in some cases different groups of developers make the software evolve in different directions, a situation which is commonly known as a fork. In the case of free software, although forking is a practice that is considered as a last resort, it is inherent to the four freedoms. This paper tries to shed some light on the practice of forking. Therefore, we have identified significant forks, several hundreds in total, and have studied them in depth. Among the issues that have been analyzed for each fork is the date when the forking occurred, the reason of the fork, and the outcome of the fork, i.e., if the original or the forking project are still developed. Our investigation shows, among other results, that forks occur in every software domain, that they have become more frequent in recent years, and that very few forks merge with the original project.

Key words: free software; open source; forks; forking; social; legal; sustainability; software evolution;

1 Introduction

Issues related to the sustainability of software projects have historically been studied in software engineering in the field of software evolution. However, research on software evolution has always implicitly assumed that development and maintenance of a software is performed by the same organization or group of developers. It is a task of the creators of the software to make it evolve [13].

But in some cases a software project evolves in parallel, lead by different development teams. This is known as "forking". The term fork is derived from the POSIX standard for operating systems: the system call used so that a process generates a copy of itself is called `fork()`. As a consequence, there exist two copies of the process that run independently and may perform different tasks. In analogy to this situation, a software fork happens when there exist two independent software projects, deriving both from the same software source code base.

Forking may happen in proprietary environments, but it is *natural* in free software as the freedom to modify a software and redistribute modifications is part of the freedoms that it grants. However, the free software movement has traditionally seen forking as something to avoid: forks split the community, introduce duplication of effort, reduce communication and may produce incompatibilities.

To the knowledge of the authors, no complete and homogeneous research on forking has been done by the software engineering research community. This paper has as main goal to raise a point of attention on this issue. The contributions of the paper can be summarized as follows: first, it offers a wide perspective of forking, identifying all significant forks. Second, it enters into detail in the reasons given for forking, presenting and classifying them. Third, it provides information on the outcome of the forks, in order to see if forking undermines the sustainability of the projects.

The structure of the paper is as follows: next, we will propose some definitions of forking and related concepts. Section 3 contains the related literature on software forking. Then, we introduce with detail the research questions that we target in this paper. Section 5 describes the methodology used for the identification and the study of forks. Results are shown in Section 6. Finally, conclusions are drawn and further research possibilities are offered.

2 Definitions

In this section we define a series of concepts used in this paper.

2.1 Clone

Cloning is the action of creating a software system designed to mimic another system. This can be done by reverse engineering or completely reimplementing from documentation or source code, or by the observation of the behavior and appearance of a software. Cloning is a common practice in free software [2].

2.2 Branch

Branching refers to the duplication of source code in a version management system. When branching occurs, parallel threads of development will take place. It is a common practice in free software projects to have branches. Software projects use branching for instance when developers do not want new features to be included in the stable branch. The popular GitHub service, which uses the `git` distributed versioning system, refers to branching as forking.

2.3 Fork

Forking occurs when a part of a development community (or a third party not related to the project) starts a completely independent line of development based on the source code basis of the project. To be considered as a fork, a project should have:

1. A new project name.
2. A branch of the software.
3. A parallel infrastructure (web site, versioning system, mailing lists, etc.).
4. And a new developer community (disjoint with the original).

2.4 Derivation

A derivation is the process of forming a new software system on the basis of an existing one, but ensuring compatibility between both systems. Derivations are common among Linux-based distributions[1]; new systems assemble software software programs and libraries that can and will be used in the other distributions without problems.

2.5 Mod

A mod is an enhancement made by enthusiasts to existing software. Mods are not standalone programs and require the user to have the original release in order to run it. They are specially popular in personal computer games. Scacchi has studied the culture of mods [19].

3 Related literature

3.1 Software engineering related research

The software engineering research literature on forking is, to the knowledge of the authors, very scarce. We have only found a technical report by Ernst *et al.* that looks specifically at forking [5]. They analyze a case study of a project forked because different requirements wanted to be met.

If we focus on software evolution articles, we find some related research. For instance, Xie *et al.* talk about parallel evolution of some free software projects in [22]. While analyzing the validity of Lehman's *laws* of software evolution on several free software projects, they discover that projects tend to have several branches (although the authors use the term *fork*) of the source code tree, one that is used for the correction of errors and another one for the inclusion of new features.

[1] A complete graph on the various families of Linux-based derivations can be found in the Wikipedia at `http://en.wikipedia.org/wiki/File:Gldt.svg`

Although forks have not been specifically the matter of research *per se*, there have been some publications where notorious forks have been used as case studies for software evolution studies. In this sense, Fischer *et al.* studied the evolution of what they call a *product family*, three variants of the BSD kernel [7]. The BSDs are studied as well by Yamamoto *et al.* to measure the similarity of the source code [23], and by Yu *et al.* to analyze their maintainability [24].

There have been some efforts to study copying of source code in free software projects. Mockus performed a massive survey of free software code and found that many projects shared files [15]. Germán *et al.* analyzed the legal consequences and issues that code being copied (*reused*) raises [10].

3.2 Related free software literature

Forking has been widely discussed outside the research literature in the free software community. In Raymond's Hacker Dictionary we can find that "[f]orking is considered a Bad Thing not merely because it implies a lot of wasted effort in the future, but because forks tend to be accompanied by a great deal of strife and acrimony between the successor groups over issues of legitimacy, succession, and design direction. There is serious social pressure against forking. As a result, major forks (such as the Gnu-Emacs/XEmacs split, the fissionings of the 386BSD group into three daughter projects, and the short-lived GCC/EGCS split) are rare enough that they are remembered individually in hacker folklore." [17].

Di Bona mentions the possibility of forking as a fear of losing control of individuals and especially companies [4]. Eric Raymond argues in [18] that the free software movement has *an elaborate but largely unadmitted set of ownership customs* that include the possibility, but mainly avoidance of forking. As Feller *et al.* put it, "[t]here is a strong taboo against forking" [6]. Neville-Neil agrees with this position and recommends to *think before you fork* [16]. He notes that only abandoned projects should be forked as developers who fork may be taken as *a petulant and spoiled child who wants to take your toys and go home.* Bezroukov indicates that ego-related issues can lead to forking, and that forking can cause the death of both initial and forked projects [1]. In Fogel's "Producing OSS" book [8], the topic is handled in detail. First, some sections are devoted to the consequences of *forkability*, what the author calls the mere possibility of doing a fork. Then, in a very practical way, he discusses how to manage a fork when it occurs.

3.3 Related research from other areas of knowledge

There are other areas of research beyond software engineering that have put some attention on forking. So, from the economics literature, Lerner and Tirole [14] identified as one of the four main roles of a leader in an FOSS project to keep the project together and prevent forking (also pointed out by Germán [9]), Weber *et al.* analyze forking from the software business perspective [21], Kogut *et al.* look it from the point of view of distributed innovation [12], and Karpf *et al.* have had some thoughts on governance, although they discuss them not using a software project but Wikipedia [11].

4 Research questions

In this study, we have targeted following research questions:

4.1 How many (significant) forks exist?

The aim of this question is to obtain the number of significant forks. When we started with this study, we had the impression that forking -being discouraged by the community- is seldom performed, but the exact number of projects that had been forked was unknown.

To answer this question we will have to identify all significant forks that have occurred in the history of free software. When identifying the forked projects, we will record the software domain in order to determine if forking is more common in certain types of software projects. Our initial (probably naive) assumption was that forking is more frequent in some domains.

4.2 Is forking becoming more frequent?

With this question we want to verify if the number of forks per year has been growing. Our initial (again naive) assumption is that forking is becoming more frequent in the last years. The first reason for assuming this is that nowadays more companies are leading free software projects; in such cases, *organizational and strategic tensions* between the goals of the company and the rest of the community may appear. If members of the community think that the company does not take their contributions and requirements into consideration, a fork may happen. The second reason is that the number of projects has grown exponentially [3], so we assume that the probability of having forks increases.

4.3 What are the main reasons for forking?

When reading the literature, and especially if we have in mind the recommendations given by the free software community, forking is a very sensitive topic. Those who create a fork have to argue that forking was a last resort [8]. With this question we want to know which are the most frequent circumstances for forking. We part from the following classification of reasons, that is derived from our knowledge of very known forks:

- Technical (addition of functionality): Some developers want to include new functionalities into the project, but the main developer(s) do not accept it. As an example of this type of fork we have `xpdf` and `Poppler`.
- More community-driven development: This occurs when the original leaders, whether a company, an institution or an independent group of developers, does not take into account the community. Then developers seeking for more open and public development practices create a fork. Examples of this type are following well-known forks: `EGCS` from `GCC`, and `XEmacs` from `GNU Emacs`.
- Discontinuation of the original project: The original project is unmaintained and a new developer community takes it over. The `Apache web server` project would be such a fork.
- Commercial strategy forks: This type of forks happens when a company forks an existing project to meet some commercial strategy. Commercial strategy forks include those where a software is released as free software by the company, or when the company creates a proprietary version of a free software. Examples of this type are `OpenOffice.org` from `StarOffice`, and `Webkit` from `KHTML`. In the opposite situation, when the community has concerns about the commercial strategy of a company, the fork belongs to the *More community-driven development* category; the `LibreOffice` fork from `OpenOffice.org` when Oracle took over Sun would be such a fork.
- Legal issues: Legal aspects such as disagreements on the license, trademarks or changes to conform laws (encryption) are included here. An example of such a fork is `X.Org`, that originated from `XFree`.
- Differences among developer team: The developer team disagrees on fundamental issues (beyond mere technical matters) related to the software development and the project. The `OpenBSD` fork from `NetBSD` is an example of such a type of fork.

4.4 What is the outcome of forking?

David A. Wheeler notes that there are four possible outcomes of a fork[2]:

1. The discontinuation of the fork.
2. A re-merging of the fork.
3. The discontinuation of the original.
4. and successful branching, typically with differentiation.

Wheeler provides an example for each of the cases, and adds that in his opinion, the discontinuation of the fork is the most common case as it is easy to declare a fork, but continuation requires considerable effort.

To Wheeler's classification we will add the situation where the original and the forked software projects have both been discontinued.

[2] `http://www.dwheeler.com/oss_fs_why.html#forking`

5 Methodology

Our intention is to document all significant forks. At first, we started performing Google searches for the term *software fork*, but the number of responses (in the range of 45,000,000) showed that this would be not embraceable.

However, one of the first terms appearing in the Google search was a page with a list of software forks in the English Wikipedia[3]. The web page contained around 30 software forks.

In addition, we have searched for *software fork* using the English Wikipedia search box. As of August 2011, the result offers 1500 Wikipedia pages. After manually inspecting all these pages, a list of 235 potential forks was obtained.

So, for the purpose of our paper, we assume that a fork is significant if a reference to it appears in the English Wikipedia. We have partially tested this assumption by looking at the 300 top-positioned results of searching for *software fork* in Google, and have not found forks we did not already have on our list.

The analysis procedure we used for each potential fork was following:

1. Locate the main website of the original software.
2. Locate the main website of the forking software.
3. Identify the software domain of the project. This information can be obtained by using the software classification of the Wikipedia, SourceForge or Free(Code).
4. Identify the reason of the fork. We inspect the website of the forking software for any information on this. In general, in its main page or in an "About" or "History" section an explanation can be found. If not, we analyze the original software web page and the Wikipedia pages of the projects for any reference. If none of the previous is successful, we perform a Google search.
5. Identify the date of the fork. Usually a date can be found together with the reason. In some cases the registration date in a software forge, the first release or the first commit in the versioning system of the forking project has been considered. The error of the dates obtained may be in the range of months, although for our purposes this is assumable.
6. Identify the outcome of the fork, including dates. Usually we have looked for the last release or the last activity on the versioning system. If there has been no activity since mid-2010, we have labeled the project as discontinued.

All this information, including any other extra relevant facts that help understanding the reason of the fork, has been written down in a log file (our research script). The research script is publicly available for replicability purposes, and for further analysis and research[4]. We have tried to indicate

[3] `http://en.wikipedia.org/wiki/List_of_software_forks`
[4] `http://gsyc.urjc.es/~grex/oss2012forking`

always the original URL where the information has been taken from. Our intention, as well, has been to use primary sources, avoiding thus Wikipedia pages, although this has not always been possible.

During August 2011 we obtained the list of significant forks. We analyzed the forks on an individual basis during a two-week period in August, and a final set of projects during one week in March[5]. The analysis was performed at a pace of 5 to 8 forks per hour.

6 Results

6.1 Number of forks

The total number of forks we have been able to identify is 220, as from the total number of potential forks (235) some of them were clones, branches or derivations. For instance, `Wireshark` appears as a fork of `Ethereal`. However, investigating this case, we found that what happened was a renaming due to trademark problems with the Ethereal name; it was the same community that was developing the software under a different name. Another example is `Gereqi`, a multimedia program that is considered in Wikipedia as a fork of `Amarok`. In its main web page, the author of `Gereqi` states that the software does not contain a single line of `Amarok`, and demonstrates it with the fact that `Gereqi` has been implemented using a different programming language.

While inspecting the list of forks, we have found projects where forking is common (e.g., `MySQL`[6], `Tux Racer`, `DC++`, `WakkaWiki`, `L2J`...). It seems that once a fork has occurred, more forks are *cheaper*, in the sense that minor reasons have to be provided to start a new fork.

Table 1 provides information of the software domains where forks have occurred. Our initial assumption was that forking was more specific to certain domains. Our results shows that this is not true, and that we have found projects forked from operating system kernels to end-user software.

We have not studied in detail if these results are proportional to the share of free software projects, for instance in Debian, Free(Code) or SourceForge. But a fast inspection of the most prominent tags at Free(Code)[7] reveals that the most popular software domains are Internet, software development, web, and multimedia, in line with our results.

Observation #1: We have found 220 significant software forks. Forking occurs in every software domain.

[5] The research script includes the dates when the forks were analyzed.
[6] See "A look at the MySQL forks", `http://lwn.net/Articles/329626/`
[7] `http://freecode.com/tags`

Software category	Frequency
Networking (servers, clients, p2p...)	52 (23.6%)
Web applications (CMS, LMS, blogs, wikis...)	34 (15.5%)
Development (IDEs, libraries...)	29 (13.2%)
Multimedia (audio, video...)	18 (8.2%)
Games/Entertainment	18 (8.2%)
System (kernel, file systems...)	17 (7.8%)
Desktop	16 (7.3%)
Utilities	12 (5.5%)
Graphics	7 (3.2%)
Databases	6 (2.7%)
Business (ERP, CRM...)	5 (2.3%)
Security	3 (1.4%)
Package Management	3 (1.4%)

Table 1. Forks by software domain.

6.2 Temporal evolution of forks

From the 220 forks, we have only identified the forking date for 210 of them. For 10 of the forks we could not obtain the date when the fork occurred, while for another 28 projects we could only find the year of forking, not the month. For the year 2011 we have only partial data, as our list of forks had been obtained August 2011.

The result of the temporal evolution of forks is shown in a histogram in Fig. 1. Prior to 1998, the number of forks is testimonial (13 in total), while the number increases in recent years, with around 20 forks per year since 2003.

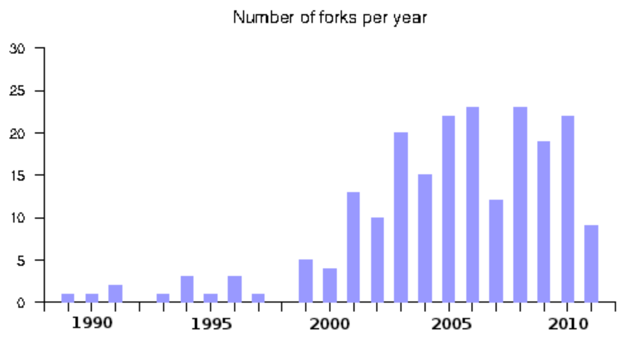

Fig. 1. Number of forks per year.

Interestingly enough, the number of forks does not follow the exponential growth path described by Deshpande and Riehle [3], so the total number of free software projects seems not to be related with the amount of forks. Maybe a proportionality is given with company-led free software projects, although this is something that we cannot state from our study and is left for further research.

Observation #2: Forks have become more frequent in recent years, but the number of forks does not grow proportionally with the number of free software projects.

6.3 Reasons for forking

The free software community is sensible about the problem of forking, as it can be seen from the fact that we have been able to find a reason for the fork for 9 out of 10 forks.

While inspecting the forks and assigning them to the categories of reasons, we included a new category -*experimental forks*- which might be seen as a subcategory of *technical* reasons. These type of forks occur when some of the developers -usually a minority- want to introduce major changes into the project, while many others prefer a more conservative approach. In this cases, instead of just opening a branch that would have it very difficult to become the future main branch, a fork is created. These types of forks could be labeled as *friendly forks*, as many of them come with the approval of the original community.

Reason	Frequency
Technical	60 (27.3%)
Discontinuation of the original project	44 (20.0%)
More community-driven development	29 (13.2%)
Legal issues	24 (10.9%)
Commercial strategy forks	20 (9.1%)
Differences among developer team	16 (7.3%)
Experimental	5 (2.3%)
Not Found	22 (10.0%)

Table 2. Main reasons for forking.

Table 2 shows the result of classifying the main reason given for the forks. We had problems to discriminate among some categories, especially between "technical", "differences among the developer team" and "more community-driven development". This is because in many of the forks two of the reasons, or even the three of them appeared in some way. So for instance, `Carrier` (formerly `Funpidgin`) was set up because the main developers of the `Pidgin` instant messaging client refused to introduce a functionality that allowed to manually resize the text input area. Although this has been sorted as a *technical* reason, a closer inspection showed that the argumentation had a more profound basis.

All in all, as observed from Table 2 although the major force for forking is of technical nature (27.3%), the main reason to fork is very distributed among the various categories under consideration.

Observation #3: For most of the forks it is possible to find the reason of forking.

6.4 Resolution of forks

We have inspected the outcome of all forks following Wheeler's (augmented) classification. The results are shown visually in Fig. 2.

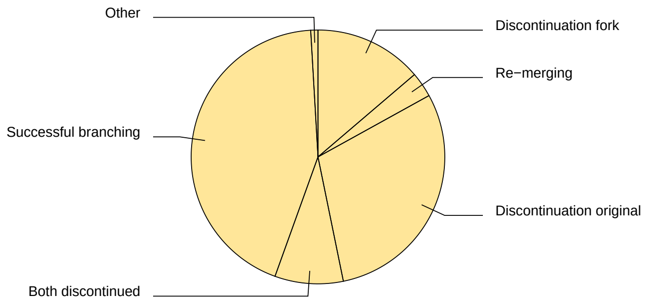

Fig. 2. Outcome of the forks.

Wheeler hypothesized that most forks do not survive their original projects. Our results throw the contrary situation: the percentage of discontinued original projects doubles the discontinued forked projects.

The results may change radically if we consider apart those forks that are a continuation of a discontinued original project. This is for instance the case for the `Apache` fork of the `NCSA HTTPd` server or the `RedDwarf` fork of the `Project Darkstar`. In both cases, the institutions behind the original projects (the NCSA and Sun, recently taken over by Oracle) ceased development and support, and a fork was created. 44 forks are a *continuation forks* (see Table 3).

These numbers show that the amount of original projects and forked projects that are discontinued is similar, being the discontinuation of the forking project slightly more frequent than the one of original projects. Nevertheless the difference is not important enough to meet Wheeler's hypothesis. We interpret this as a consequence of developers being aware of the implications of forking, including the disadvantages of competing with the original

Outcome	Frequency
Successful branching, typically with differentiation	95 (43.6%)
The discontinuation of the original	65 (29.8%)
Fork was response to (possible or actual) discontinuation of original	44 (20.0%)
Fork was due to other reasons	21 (9.8%)
The discontinuation of the fork	30 (13.8%)
None of them currently under development	19 (8.7%)
A re-merging of the fork	7 (3.2%)
Other	2 (0.9%)

Table 3. Outcome of forks, considering if the cause of the fork is the possible or actual discontinuation of the original project.

project (the need for making the fork reasonable to the community, the use of a new name for the forked software project, etc.). Only if they consider that they have enough reasons and good changes of success, a fork is done.

From Table 3, we can also interpret that when forking occurs the sustainability of the software is ensured. Our argument to sustain such a claim is that forked projects survives for long time - whether original or as the fork. The number of projects where both, original and forked, projects are discontinued is very low compared to the usual numbers of abandoned projects in free software [20].

The number of forks that merge with the original project is very low. In addition, merging often happens by dismissing one of the projects as developers report that integrating changes is not a simple task. In the case of `GCC` and `EGCS`, the technical superiority of the fork made it become the *official* compiler of the GNU project, discontinuing the `GCC` source code base. The opposite happened with the `libc` Linux fork from `GNU libc`; with a new, enhanced version of `GNU libc`, Linux stopped using its own `libc`. The authors report that "changes that had been made in Linux libc could not be merged back into glibc because the authorship status of that code was unclear and the GNU project is quite strict about recording copyright"[8].

Observation #4: Most of the forks evolve in parallel to the original project. The chance of discontinuation of a fork is almost the same as the original, even if they have a disadvantageous starting situation. Software that forks ensures sustainability. Few forks re-merge, and when this happens, fewer integrate source code.

7 Conclusions and further research

In this paper we have presented to the knowledge of the authors the first comprehensive study on software forks. We have provided some insight into

[8] http://blogs.fsfe.org/ciaran/?p=85

the temporal patterns of software forks, the reasons for forking and what the outcome of the forking has been for the original and forking project.

In the opinion of the authors, forking is going to be a more relevant and frequent situation in the next future. Free software is nowadays already the root of many other software project which build on top of it (for instance Android); the disparity in goals among the root project and others building on top of it will produce more forks.

On the other hand, technology is lowering the technical barrier of creating a fork. For instance, distributed versioning systems such as `git` are more suitable to forking, as they provide a copy of the complete history of the project; not only a branch can be built, but a complete copy of the infrastructure is provided. However this is still not the case for bug-tracking systems and mailing lists, which are not distributed and difficult to fork. In our opinion, technology should make forks easier; the convenience of forking should just be a strategic matter that allows to maintain balances among the stakeholders of a project. On the other hand, lowering the technological barrier to fork may increase the number of *friendly* experimental forks that boost innovation.

There are several topics related to forking that further research should target. First, it would be interesting to find out how much original and forking projects collaborate, even in an involuntary way, by exchanging code, bug reports and fixes. Second, parallel software development should be devoted a closer look; technology and processes should facilitate integration. Third, further research should focus on understanding how the community moves when a fork occurs. This would include among others answering questions such as where the key developers go, how many of the developers are active both in the original and forking projects, or if there is any correlation between a *positive resolution* of the fork and who pushed for it. Fourth, we think that it would be valuable to focus on how the socioeconomic and technical context may influence the probability and the type of a fork. So, for instance, if project maturity boosts forking or not. And fifth, we think that a study on forking in projects led by software companies could answer many interesting questions that have not been addressed in our work.

8 Acknowledgements

The work of G. Robles and J.M. González-Barahona has been funded in part by the European Commission under project ALERT (FP7-IST-25809) and by the Spanish Gov. under project SobreSale (TIN2011-28110). We would like to thank the anonymous reviewers for their insightful comments.

References

1. Nikolai Bezroukov. A second look at the Cathedral and The Bazar. *First Monday*, 4(12), 1997.

2. Andrea Capiluppi, Cornelia Boldyreff, and Klaas-Jan Stol. Successful reuse of software components: A report from the open source perspective. In *International Conference on Open Source Software Systems*, pages 159–176, 2011.
3. Amit Deshpande and Dirk Riehle. The Total Growth of Open Source. In *Proceedings of the Fourth Conference on Open Source Systems*, page 197209. Springer Verlag, 2008.
4. C. DiBona, S. Ockman, and M. Stone. Introduction. In C. DiBona, S. Ockman, and M. Stone, editors, *Open Sources. Voices from the Open Source Revolution*, pages 1–17. O'Reilly, 1999.
5. Neil A. Ernst, Steve M. Easterbrook, and John Mylopoulos. Code forking in open-source software: a requirements perspective. *CoRR*, abs/1004.2889, 2010.
6. J. Feller and B. Fitzgerald. A Framework Analysis of the Open Source Software Development Paradigm. In *21st International Conference of Information Systems*, pages 58–69, Brisbane, Australia, 2000.
7. Michael Fischer, Johann Oberleitner, Jacek Ratzinger, and Harald Gall. Mining evolution data of a product family. *ACM SIGSOFT Software Engineering Notes*, 30(4):1–5, 2005.
8. Karl Fogel. *Producing Open Source Software - How to run a successful free software project.* O'Reilly, 2005.
9. Daniel M. Germán. The GNOME project: a case study of open source, global software development. *Journal of Software Process: Improvement and Practice*, 8(4):201–215, 2004.
10. Daniel M. Germán, Massimiliano Di Penta, Yann-Gal Guéhéneuc, and Giuliano Antoniol. Code siblings: Technical and legal implications of copying code between applications. In *International Working Conference on Mining Software Repositories (MSR)*, pages 81–90. IEEE, 2009.
11. David Karpf. What Can Wikipedia Tell Us about Open Source Politics? In *Conference Proceedings of JITP 2010: The Politics of Open Source*, pages 2–31, 2010.
12. Bruce Kogut and Anca Metiu. Open-source software development and distributed innovation. *Oxford Review of Economic Policy*, 17(2), 2001.
13. Manny M. Lehman and L. A. Belady, editors. *Program evolution: Processes of software change.* Academic Press Professional, Inc., San Diego, CA, USA, 1985.
14. Josh Lerner and Jean Tirole. Some simple economics of open source. *Journal of Industrial Economics*, 50(2):197–234, 2002.
15. Audris Mockus. Amassing and indexing a large sample of version control systems: Towards the census of public source code history. In *MSR '09: Proceedings of the 2009 International Working Conference on Mining Software Repositories*, pages 11–20, 2009.
16. George V. Neville-Neil. Think before you fork. *Commun. ACM*, 54:34–35, June 2011.
17. Eric S. Raymond. *The New Hacker's Dictionary.* MIT Press, Cambridge, MA, USA, 3rd edition, 1996.
18. Eric S. Raymond. Homesteading the Noosphere. In Eric S. Raymond, editor, *The Cathedral and the Bazaar. Musings on Linux and Open Source by an Accidental Revolutionary*, pages 79–135. O'Reilly, 1999.
19. Walt Scacchi. Computer game mods, modders, modding, and the mod scene. *First Monday*, 15(5), 2010.

20. Charles M Schweik, Robert English, Qimti Paienjton, and Sandy Haire. Success and Abandonment in Open Source Commons: Selected Findings from an Empirical Study of Sourceforge.net Projects. In *Proceedings of the Sixth International Conference on Open Source Systems 2010*, pages 91–101, 2010.
21. Steven Weber. *The Success of Open Source*. Harvard University Press, April 2004.
22. Guowu Xie, Jianbo Chen, and Iulian Neamtiu. Towards a better understanding of software evolution: An empirical study on open source software. In *Proceedings of the Internacional Conference on Software Maintenance*, pages 51–60. IEEE, 2009.
23. Tetsuo Yamamoto, Makoto Matsushita, Toshihiro Kamiya, and Katsuro Inoue. Measuring similarity of large software systems based on source code correspondence. In Frank Bomarius and Seija Komi-Sirvi, editors, *PROFES*, volume 3547 of *Lecture Notes in Computer Science*, pages 530–544. Springer, 2005.
24. Liguo Yu, Stephen R. Schach, Kai Chen, Gillian Z. Heller, and A. Jefferson Offutt. Maintainability of the kernels of open-source operating systems: A comparison of Linux with FreeBSD, NetBSD, and OpenBSD. *Journal of Systems and Software*, 79(6):807–815, 2006.