PERSONAL VERSION

CODE FORKING, GOVERNANCE, AND SUSTAINABILITY IN OPEN SOURCE SOFTWARE

Linus Nyman Post-doctoral, Information Systems Science, Hanken School of Economics

Juho Lindman Assistant professor, Information Systems Science, Hanken School of Economics juho.lindman@hanken.fi

> The ability to fork code – a central freedom of open source software – is what keeps communities vibrant and companies honest.
>
> Glyn Moody
> Technology writer and journalist

Abstract

The right to fork open source code is at the core of open source licensing. All open source licenses grant the right to fork their code, that is to start a new development effort using an existing code as its base. Thus, code forking represents the single greatest tool available for guaranteeing sustainability in open source software. In addition to bolstering program sustainability, code forking directly affects the governance of open source initiatives. Forking, and even the mere possibility of forking code, affects the governance and sustainability of open source initiatives on three distinct levels: software, community, and ecosystem. On the software level, the right to fork makes planned obsolescence, versioning, vendor lock-in, end-of-support issues, and similar initiatives all but impossible to implement. On the community level, forking impacts both sustainability and governance through the power it grants the community to safeguard against unfavourable actions by corporations or project leaders. On the business-ecosystem level forking can serve as a catalyst for innovation while simultaneously promoting better quality software through natural selection. Thus, forking helps keep open source initiatives relevant and presents opportunities for the development and commercialization of current and abandoned programs.

1 INTRODUCTION

This article addresses the question of how the right to fork open source projects – to use the source code of an existing program to start a new, independent version – works as a governance mechanism to provide sustainability in open source software. The concept of sustainability is under debate, with numerous rubrics against which the sustainability of a product may be measured (e.g., Connelly, 2007; **tinyurl.com/atjcgq3**; Davison, 2001; **tinyurl.com/aukl5ch**; McManus, 1996; **tinyurl.com/a5usfo3**). Within the context of the current study, sustainability is defined as the possibility of an open source program to continue to serve the needs of its developers and users.

While code forking may lead to redundant independent efforts, it represents the single greatest tool available for guaranteeing sustainability in open source software. In this article, we examine code forking within open source initiatives and discuss the managerial implications of code forking. The article is structured as follows: first, we offer some background on code forking; second, we look at how code forking affects governance on the three levels mentioned; finally, we explain the relevance of these findings and their management implications.

1.1 Background

Code forking has often been viewed in a negative light. At the core of this negative view is the continued use of a restrictive, and perhaps outdated, definition of the term forking. Until recently, the term fork was mainly used to describe a situation in which a developer community had split into competing camps, each continuing work on their own, incompatible version of the software (see, for example, Raymond, 1999; **tinyurl.com/3ald3**; Fogel, 2006; **tinyurl.com/3dx2py**). Hence, the negative tone found in discussions of forking has been related to concerns regarding the hindered progress, wasted resources, and potential demise of one or both of the projects. In recent years, the term forking has come to be used in a much broader context, encompassing all cases in which one takes an existing code base and implements it in a separate project (see, for instance, GitHub; **tinyurl.com/7uc94sk**). In the context of this study, we adhere to this broader definition of forking.

While there are many reasons why projects are forked, the most common reason is the desire to modify the original program to better address a specific need (Nyman and

Mikkonen, 2011; **tinyurl.com/arntyur**). Forks may also be planned, temporary divergences intended to test new ideas and features, with the intention of later integrating effective improvements back into the original (Nyman and Mikkonen, 2011; **tinyurl.com/arntyur**; see also GitHub; **tinyurl.com/7uc94sk**). The right to fork code is built into the very definition of what it means to be an open source program. The third criteria of the Open Source Initiative's (OSI; **opensource.org/osd.html**) definition of open source states that the license "must allow modifications and derived works." Similarly, the Free Software Foundation's Free Software Definition (FSD; **gnu.org/philosophy/free-sw.html**) states that users have the freedom to "run, copy, distribute, study, change and improve the software." All spinoff initiatives can be considered forks as they are "modified or derived" (OSI) or "copied, changed and improved". The possibility of forking any project affects the governance and sustainability of all open source programs.

Software is editable, interactive, reprogrammable, distributed, and open (Kallinikos et al., 2010; **tinyurl.com/4zn6cun**). These characteristics dictate that software is prone to being changed, repaired, and updated rather than remaining fixed from the early stages of the design process. The openness combined with the granular composition of the software offer new ways of governance (Benkler, 2006; **tinyurl.com/6ftot3**). This governance is not tied to over-appropriating a natural resource (Ostrom, 1991; **tinyurl.com/b8rc2pu**), but rather related to ways in which a group of developers, following institutional rules, collectively produce a public good (Schweik et al., 2010; **tinyurl.com/aqxy2jp**).

2 THREE LEVELS OF GOVERNANCE

2.1 Software level

The nature of the industry dictates that programs cannot maintain a stable steady state for an extended period of time. They must continue to evolve in order to remain useful and relevant. Without continual adaptation, a program will progressively become less satisfactory (Lehman, 1980; **tinyurl.com/b2mpkw3**). Conversely, truly successful software is able to adapt and even outlive the hardware for which it was originally written (Brooks, 1975; **tinyurl.com/awg3rrw**). Therefore, the ability to change and evolve is a key component of software sustainability. Although stagnation may be a precursor to obsolescence, obsolescence need not creep into a project over time; it is often a design feature.

Popularized in the 1950s by American industrial designer Brooks Stevens (The Economist, 2009; **tinyurl.com/ahws66g)**, the concept of planned obsolescence stands in stark contrast to the concept of sustainability. Stevens defined planned obsolescence as the act of instilling in the buyer "the desire to own something a little newer, a little better, a little sooner than is necessary" (Brooks Stevens' biography; **tinyurl.com/bbs8a3c**). Considered "an engine of technological progress" by some (Fishman et al., 1993; **tinyurl.com/bye2n5r**), yet increasingly problematized in the business ethics literature (Guiltinan, 2009; **tinyurl.com/alr2c92**), planned obsolescence is part of every consumer's life. Although contemporary software development and distribution have characteristics that differ substantially from the industrial products of the 1950s, the revenue models of companies in the software marketplace often welcome elements such as system versioning, to encourage repurchases of a newer version of the same system, or vendor lock-ins that limit the customer choice to certain providers of system or product (for a further review, see Combs, 2000; **tinyurl.com/aq2wl7h**). Newer versions of programs may introduce compatibility problems with earlier operating systems or programs (e.g., lack of backwards compatibility in Internet Explorer, Microsoft Office, or OS X's OpenStep APIs). Some programs also introduce new file formats, which can cause compatibility issues with earlier versions of the program (e.g., docx vs. doc). Furthermore, end-of-life announcements and concerns over end-of-support deadlines may encourage users to upgrade, regardless of the real need to do so.

The right to fork code makes implementing such elements impracticable in open source. The right to improve a program, the right to combine many programs, and the right to make a program compatible with other programs and versions are all fundamental rights that are built into the very definition of open source. Research has shown these rights are often exercised (Fitzgerald, 2006; **tinyurl.com/al995aj**). The result of this constant collaborative improvement in open source systems is that any program with the support of the open source community can enjoy assured relevance rather than planned obsolescence. Furthermore, with renewed community interest, programs that have decayed and fallen into disuse can be revived and updated by forking the code from the original program. In fact, this is a fairly common practice: of the almost 400 forks studied by Nyman and Mikkonen (2011; **tinyurl.com/arntyur**), 7% involved the reviving of an abandoned project. As long as there is sufficient community interest in a project, forking can allow for constant improvement in software functionality.

2.2 Community level

The possibility to fork is central to the governance of any open source community. The shared ownership of open source projects allows anyone to fork a project at any time. Therefore, no one person or group has a "magical hold" over the project (Fogel, 2006; **tinyurl.com/ahbh8nt**). Since a fork involving a split of the community can hurt overall productivity, Fogel notes that the potential to fork a program is "the indispensable ingredient that binds developers together".

One of the concerns among open source communities is what Lerner and Tirole (2002; **tinyurl.com/bfmaxl4**) call the hijacking of the code. Hijacking occurs when a commercial vendor attempts to privatize a project's source code. The 2008 acquisition of MySQL, (**mysql.com**), an open source relational database management system, by Sun Microsystems and subsequent acquisition of Sun by Oracle is an example of a case involving community concern over potential hijacking. It had been argued that such a series of acquisitions would lead to the collapse of both MySQL and the open source movement at large (Foremski, 2006; **tinyurl.com/yesjhw7**). Responding to such claims, Moody (2009; **tinyurl.com/cbrq7g**) noted that, while open source companies can be bought, open source communities cannot. Forking provides the community that supports an open source project with a way to spin off their own version of the project in case of such an acquisition. Indeed, this is what happened in the case of MYSQL. The

original MySQL developer, Michael (“Monty”) Widenius, forked the MySQL code and started a new version under a different name, MariaDB, due to concerns regarding the governance and future openness of the MySQL code (for details, see Widenius' blog [February 5, 2009; **tinyurl.com/btr9bm6** and December 12, 2009; **tinyurl.com/ba58vpp**] and press release (**tinyurl.com/auvaxbn**)).

Similarly, in 2010, community concerns regarding governance led to a forking of the OpenOffice (OO; **openoffice.org**) project. The Document Foundation, which included a team of long-term contributors to OO, forked the OO code to begin LibreOffice; (**libreoffice.org**). The spinoff project emphasized the importance of a “transparent, collaborative, and inclusive” government (The Document Foundation; **tinyurl.com/bzmw5p2**). A recent analysis of the LibreOffice project indicates that this fork has resulted in a sustainable community with no signs of stagnation (Gamalielsson and Lundell, 2012; **tinyurl.com/a9ev4hu**). Given that forking ensures that any project can continue as long as there is sufficient community interest, we have previously described forking as the “invisible hand of sustainability” in open source software (Nyman et al., 2011; **tinyurl.com/b8bzorg**).

Commonly, forking occurs due to a community’s desire to create different functionality or focus the project in a new direction. Such forks are based on a difference in software requirements or focus, rather than a distrust of the project leaders. When they address disparate community needs, different versions can prosper.

In a traditional company, it is the management, headed by the CEO and board of directors, that controls the company and provides the impetus for continued development. While the vision of the leadership is similarly integral to the eventual success of any open source project, their continued control is more fragile and hinges upon their relationship with and responses to the community. Forking cannot be prevented by business models or governance systems. The key lies in appropriate resource allocation and careful community management. Managers must strike a delicate balance between providing a driving force while appeasing and unifying the community. (For an overview of open source governance models, see OSS Watch; **tinyurl.com/bjqpnkn** for discussion on building technical communities, see Skerrett, 2008; **timreview.ca/article/160**; for discussion on open source community management, see Byron, 2009; [**timreview.ca/article/258**]).

2.3 Business-ecosystem level

Within the dynamic world of open source software, natural selection acts as a culling force, constantly choosing only the fittest code to survive (Torvalds, 2001; **tinyurl.com/aaxqux7**). However, the right to fork means that any company can duplicate any competitor's open source software distributions; thus, competitive advantage cannot depend on the quality of the code alone. However, it is worth stressing that possibility does not equal success. The right to fork a commercially successful program with the intention of competing for the same customer base still leaves the would-be competitor with issues regarding trademarks, brand value and recognition, as well as the existing developer and user base of the original program. Even though forking allows companies to compete with identical open source software, it is nevertheless cooperation that is considered to be the key to corporate success (Skerrett, 2011; **timreview.ca/article/409**; Muegge, 2011; **timreview.ca/article/495)**.

Open source software is free, but it is also increasingly developed and supported for commercial gains (Wheeler, 2009; **timreview.ca/article/229**). While the right to fork may seem to make for a harsh business environment, open source companies can and do thrive. With its billion-dollar revenue, (**tinyurl.com/b7py36u**) Red Hat is one such example. While their revenue primarily comes from subscriptions and services related to their software (see Suehle's [2012; **timreview.ca/article/513**] TIM Review Q&A for a more in-depth look at the secret of Red Hat's success), Red Hat's programs themselves are largely based on forks of programs by other developers. This phenomenon of combining forked programs is not unique to Red Hat: the hundreds of different Linux distributions (**tinyurl.com/85r9o**) are all made possible by the forking of existing products and repackaging them as a new release.

Forking lays the building blocks for innovators to introduce new functionalities into the market, and the plethora of online forges have hundreds of thousands of programs available for forking and reuse in any new, creative way the user can imagine, allowing for the rapid adaptation to the needs of end users. Hence, the practice of forking allows for the development of a robust, responsive software ecosystem that is able to meet an abundance of demands (Nyman et al., 2012; **tinyurl.com/acg3fp2**).

The old adage, "one man's trash is another man's treasure" is particularly salient in open source software development. Soon after Nokia's abandonment of the MeeGo

project in 2011 (press release; **tinyurl.com/ad5lh6b**; MeeGo summary; **tinyurl.com/9u4xrno**), the Finnish company Jolla announced that it would create a business around its revival, made possible by forking the original code (press release; **tinyurl.com/7bzbo9h**). On July 16, 2012, Jolla announced a contract with D. Phone, one of the largest cell phone retailers in China, and on November 21 they launched Sailfish OS (**tinyurl.com/a4yot8h**). However, one does not need to be an open source business to benefit from the right to fork. Forking can also aid companies who choose to use an existing program, or develop it for personal use. The requirement in open source to share one's source code is linked with distribution, not modification, which means that one can fork a program and modify it for in-house use without having to supply the code to others. However, a working knowledge of licenses as well as license compatibility (when combining programs) is crucial before undertaking such an endeavour (for a discussion of licenses, see St. Laurent [2004; **tinyurl.com/befxwvc**], Välimäki [2005; **tinyurl.com/ahljzwu**], or Meeker [2008; **tinyurl.com/am93qol**] for a discussion of architectural design practices in the combining of licenses, see Hammouda and colleagues [2010; **tinyurl.com/bfp82mw**].

A summary of the ways in which forking can affect governance and help ensure sustainability is provided in Table 1.

Table 1 Forking and its effect on governance

Level	How Forking Provides Sustainability	Examples
Software	The right to fork protects against planned obsolescence, versioning, and vendor lock-in Disuse due to decay can be countered by forking and updating	Microsoft Word vs. LibreOffice Fairly common open source practice (for examples, see Nyman [2011;**tinyurl.com/arntyur**])
Community	Prevents hijacking and other unfavourable actions by project leaders or owners through giving developers the option to continue their own version of the program	MariaDB forked from MySQL, LibreOffice forked from OpenOffice
Ecosystem	Increases innovative potential by allowing for the combination and modification of open source projects Abandoned (or badly handled) projects can be revived, creating new business opportunities	Plethora of different Linux distributions Abandoned) MeeGo forked to create Sailfish

Managerial Implications

Managers should consider the following implications of code forking:

- An abandoned project can become a business opportunity.
- Neither business models nor governance systems can completely prevent forking. Thus, developer and community satisfaction is of key importance.
- A strong, vibrant community is a key issue to consider when implementing an open source program. When acquiring systems, the potential of forking in open source software – in particular when coupled with a strong community – provides opportunities to avoid versioning and vendor lock-in to one provider of a product or system. However, while community is important, it is not the only factor to consider. For more on evaluating and selecting open source software for corporate use, see the May 2008 issue of TIM Review, including topical articles by Golden (2008; (**timreview.ca/article/145**), von Rotz (2008; (**timreview.ca/article/147**), and Semeteys (2008; (**timreview.ca/article/146**).
- There are thousands of open source programs already in existence, which can be forked. If a need for software arises and open source is an option, begin by analyzing what already exists on code repositories such as SourceForge; (**sourceforge.net**) and GitHub; (**github.com**). Keep in mind that it is distribution, not modification, that obligates the sharing of the source code. Be sure to read up on licenses first!

3 CONCLUSION

Forking sits at the intersection of several different open source topics, such as software development, governance, and company participation in communities and business ecosystems. In the interest of clarity, we have simplified the categorization of the multifaceted concept of forking. In actuality, there is overlap among the categories: a strong community offers better insurance of sustainability of the software level, while better software can more easily attract a bigger community. Both a poorly handled community and an abandoned project can spawn a business ecosystem competitor.

The right to fork code is intrinsic to open source software and is guaranteed by all open source licenses. This right to fork has a significant effect on governance and helps ensure the sustainability of open source software. We have analyzed the effect of forking on three different levels: the software level, the community level, and the ecosystem level. On a software level, code forking serves as a governance mechanism for sustainability by offering a way to overcome planned obsolescence and decay, as well as versioning, lock-in, and related concerns. On a community level, code forking ensures sustainability by providing the community with an escape hatch: the right to start a new version of the program. Finally, on an ecosystem level, forking serves as a core component of natural selection and as a catalyst for innovation. Online forges offer a plethora of publically available programs that can serve as the building blocks of a new creation. Current projects can be forked, abandoned projects can be revived and commercialized, or programs can be combined in novel ways to better meet the needs of both the developers and end users. It is the right to fork that moulds the governance of open source projects and provides the dynamic vigour found in open source computing today.