Forking: the Invisible Hand of Sustainability in Open Source Software

Linus Nyman[1], Tommi Mikkonen[2], Juho Lindman[1], and Martin Fougère[1]

1 Hanken School of Economics, Helsinki, Finland
firstname.lastname@hanken.fi

2 Tampere University of Technology, Tampere, Finland
tommi.mikkonen@tut.fi

Abstract. The ability to create and maintain high-quality software artifacts that preserve their usability over time is one of the most essential characteristics of the software business. In such a setting, open source software offers excellent examples of sustainability. In particular, safeguarding mechanisms against planned obsolescence by any single actor are built into the very definition of open source development. The most powerful of these safeguarding mechanisms is the ability to fork the project as a whole. In this position paper, we argue that the possibility to fork any open source program serves as the invisible hand of sustainability, ensuring that the code can always remain open and that the code that best fulfills the needs of the community will live on.

1 Introduction

Sustainability is a concept which is often automatically associated with open source software. Indeed, access to the source code enables developers to build solutions that are better protected from the actions of any single developer, organization, or company associated with the software. The openness of the source code also guarantees that decisions concerning the software artifact enjoy a measure of transparency.

In this position paper we address the role of code forking – a situation in which several versions of a piece of software originating from a single, shared code base are developed separately – in ensuring the long-term sustainability of a software system. Furthermore, we advocate the freedom that developers have to create novel features that may well go beyond what the developers who began the project originally expected or planned. This freedom will also nurture open source projects through difficult times and extreme events that could otherwise prove lethal, such as hostile commercial acquisitions which may cause changes in the practices of the project.

The rest of the paper is structured as follows. In Section 2, we briefly address the sustainability of digital objects, focusing on open source and planned obsolescence, which is generally associated with almost all of the systems in common use. In Section 3, we discuss code forking, which can serve as an element that supports long-term sustainability. In Section 4, we offer our view of the effect code forking has on sustainability. Finally, in Section 5, we draw some of the main conclusions.

2 Sustainability and planned obsolescence

The sustainability of a product can be interpreted in many ways. From the viewpoint of the consumer, there are at least two central elements: quality and staying power. In other words, we often seek a high-quality product which remains usable for as long as possible.

This view of sustainability contrasts with what is known as "planned obsolescence", a term popularized in the 1950s by American industrial designer Brooks Stevens [1]. 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" [2]. From the fashion industry, where last year's models are designed to look out-of-date by the time this year's models come around, to the software industry, where the norm is for software to be compatible with older models, but not with newer ones, planned obsolescence has become an inescapable part of the consumer's everyday life.

Of course, digital artifacts differ substantially from the end products of 1950s industrial design, or even those of today. The main differences are related to their characteristics as editable, interactive, reprogrammable, distributed, and open [4]. These characteristics dictate that for example, software as an artifact is prone to being changed, repaired and updated rather than remaining fixed from the early stages of the design process. The software marketplace has transferred planned obsolescence to the digital realm by creating ways to benefit from these artifact characteristics. The revenue models of companies that operate in the software marketplace thus welcome versioning, lock-ins, systems competition, and network effects [6].

Open source software offers an alternative to some of the pitfalls of planned obsolescence. Rather than needing to buy something "a little newer, a little better", the open source community can simply make the existing product a little – or a lot – newer and better. In open source, anything, once invented, once written, need never be rewritten. On the other hand, the software product is never ready, but can become stable and mature enough for the developer community. If the community interest is there, the software can always be improved.

The right to improve a program, the right to make it portable to newer as well as older programs and versions, and the right to combine many programs into an even better whole are all rights built into the very definition of open source. The net result is that, in open source systems, any program which has the support of the open source community will enjoy assured relevance rather than planned obsolescence.

In fact, planned obsolescence in open source is impossible to implement due to a practice which is at once both the sustainer and potential destroyer of open source programs: the code fork.

3 Code forking

A popular metaphor in economics is Adam Smith’s “invisible hand” which guides the marketplace. We claim that open source software has its own invisible hand: the fork. In fact, even the *possibility* of a fork – something which is guaranteed by all open source licenses – usually suffices. Actual forks are rare, but it is enough that they *could* happen, should the conditions in which the project is being developed change radically. In recent years, examples of using a fork for the sustainability of a community include high-profile cases such as the forking of OpenOffice (http://www.openoffice.org/) into LibreOffice (http://www.libreoffice.org/) and the creation of various projects from the code base of MySQL (http://www.mysql.com/).

A broad definition of a code fork is when the code from an existing program serves as the basis for a new version. This can be the result of a split in the developer community regarding the software artifact, its development practice, or the direction of the development, and is then usually followed by a split in the user community. Code forking in open source software is paradoxical in nature; it is simultaneously both one of the greatest threats an individual project faces, as well as the ultimate sustainer: insurance that, as long as users find a program useful, the program will continue to exist.

The threat to the program comes mainly in the form of the (potential) dilution of both users and developers. As Fogel [3] has noted, it is not the existence of a fork that hurts a project, but rather the loss of developers and users. The benefits of a fork come in ensuring that the program can continue to exist regardless of external circumstances. If, for instance, the developers of a program under a permissive license decide to relicense it under either a proprietary or otherwise less favorable license, the community can fork a new version and continue development. In the early days of open source, forking enabled the community to choose which version of UNIX to adopt. Forks can also serve as an escape hatch for projects and developers who find themselves cornered or unable to continue on a planned course.

In the case of a program remaining under an open source license, but where the people or company shepherding the code make decisions which run counter to the interests of the larger community and developers, code forking ensures the continued development of the code, as the community and developers can fork a new version on which to continue working. Even situations in which different versions of a program fork live on can benefit one another, as one community can incorporate into their program anything developed by the other community.

4 Code forking and sustainability

With open source, one can always fork a project; the code is available for download and the open source licensing terms impose no conditions which would in any way require developers to adhere to the original development line. In successful projects, however, a balance of power seems to exist where developers are happy enough to follow the project leader as long as the project leader listens to developers' views enough to keep them onboard. This balance creates continuity for long-term cooperation.

The mere possibility of forking has a huge impact on how open source programs are governed and developed [3], and provides the community with the tools it needs to handle situations in which a program could become obsolete. This can happen for numerous reasons, including the creation of a new version of the system, a change in licensing, porting to a new hardware environment, a change in program focus, and so forth.

For an open source project to remain sustainable, it must evolve with its users. Code forking and, indeed, the mere possibility of forking, is one of the key factors that ensures that open source will continue to evolve and thus remain sustainable. Open source programs can also cease to develop; some programs and pieces of code live on while others die out. Forking, as well as the effect of the possibility of forking, ensures that the selection lies in the hands of the community itself, which is perhaps the greatest guarantee of sustainability one could possibly ask for. At its best, open source software, guided by the invisible hand of forking, may well render obsolescence itself obsolete.

5 Conclusions

In this position paper, we argue that forking has the capability of serving as an invisible hand of sustainability that helps open source projects to survive extreme events such as commercial acquisitions, which may dramatically affect licensing and community support practices. While forking can occur for numerous reasons, some of which are less dramatic than others (see [5] for a survey of SourceForge projects), the mere possibility of forking is a powerful incentive for ensuring continuity.

To summarize, we claim this invisible hand is an essential element for the long-term viability of a project's development and thus the sustainability of the resulting open source software artifacts, and that without the opportunity to fork, many events now often considered mere annoyances could lead to the termination of a project.

References

[1] Planned obsolescence, The Economist, 23 March 2009. Available at: http://www.economist.com/node/13354332, accessed 14 September 2011.

[2] Brooks Stevens biography, available at: http://www.brooksstevenshistory.com/brooks_bio.pdf, accessed 14 September 2011

[3] Fogel (2006) Producing Open Source Software. O'Reilly, Sebastopol, CA.

[4] Kallinikos, J., Aaltonen, A., and Attila. M. (2010). A theory of digital objects. First Monday, Volume 15, Number 6-7 June 2010.

[5] Nyman, L. and Mikkonen, T. (2011) To Fork or Not to Fork: Fork: Motivations in SourceForge Projects. Proceedings of the 7th International Conference on Open Source Systems (OSS 2011), 259-268, Springer.

[6] Shapiro, C., and Varian, H. (1998). Information Rules: A Strategic Guide to the Network Economy. Boston, MA: Harvard Business School Press.