Chapter 7
To Fork or Not to Fork: Fork Motivations in SourceForge Projects

Linus Nyman
Hanken School of Economics, Finland

Tommi Mikkonen
Tampere University of Technology, Finland

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. Although forking in open source software does not require the permission of the original authors, the new version 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. The authors present the results of a study of forks performed in SourceForge (http://sourceforge.net/) and list the developers' motivations for their actions.

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. In open source software development 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/) and the creation of various projects from the code base of MySQL (http://www.mysql.com/), the whole concept of forking has seen little study. Furthermore, developers' motivations for forking are understood even less, although at times it seems

DOI: 10.4018/978-1-4666-2937-0.ch007

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: First, the paper discusses the necessary background for explaining some of the technical aspects associated with forking, and then we introduce the fashion in which the research was carried out. Next we offer insight into our most important findings, and discuss them in more detail. We then propose some directions for future research, and conclude the paper with some final remarks.

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 (Fogel, 2006). 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 (2001) 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 (2011) 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 (2006): 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 (2004) and Fogel (2006) 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