Social Forking in Open Source Software: An Empirical Study

Kam Hay Fung[1], Aybüke Aurum[1], David Tang[1]

[1]School of Information Systems, Technology and Management,
The University of New South Wales, Australia

kamhayfung@ieee.org, aybuke@unsw.edu.au, dtang@atlassian.com

Abstract. Forking is the creation of a new software project by making a copy of artefacts from another project. Forking is gaining traction in industry because of the maturity of distributed version control systems and the abundance of open source software (OSS) and hosting platforms that support forking. However, forking in OSS is a poorly understood practice in research, often assumed to be damaging to the open source community. This research aims to explore social forking. It uses a conceptual model for forking centring on three key concepts - forks (i.e. created projects), communities (i.e. groups of forks) and contributions (i.e. changes contributed from a forked project to the project from which its artefacts were copied) - to empirically analyse nine public domain JavaScript development communities in GitHub, a web site for hosting social coding. The analysis examined the relationships of these communities, the nature of forking, and the way in which forking and contributions were used in a social setting.

Keywords: forking, cloning, open source software, social coding

1 Introduction

The open source software (OSS) initiative has provided an invaluable source of learning for the software industry, running counter to many existing theories and explanations [20], and offering practices and characteristics in contrast to commercial software development. While much of the open source landscape has been explored – in terms of participants' motivations [8], social and technical characteristics of projects, management issues, legal issues, adoption and business models – a changing technological milieu is likely to offer new opportunities for learning, as it nudges the open source community in subtle but interesting ways. This is because OSS development, as a geographically distributed and asynchronous process, is largely mediated by Internet technologies such as mailing lists, wikis, bug trackers and version control systems [1] – which, as with any technology, evolve over time.

Distributed revision control systems (DRCS) [13] provide the infrastructure and platforms for hosting OSS projects. DRCS, as distinct from the traditional, centralised revision control systems, enable the sharing of code between peers without communi-

cating through a central server [9]. These platforms encourage developers to *fork* each other's projects [4] – that is, to *copy* and *publish* an OSS's code from repositories owned and maintained by others to make changes. These changes can be promoted to the original OSS (e.g. as enhancements) or used to manifest it into a different OSS. The social phenomenon of forking refers to the heedful interaction of members in an OSS community, via forking, in driving the advancement of the OSS.

We do believe that forking has its importance and practicality in OSS development. It, however, has received mixed reception in the literature and which is often contingent on anecdotes and conjecture [4]. The lack of interest in and in-depth research on this topic also stems from the fact that forking in OSS is considered to be unlikely to occur [15]. This brings us to our research question:

How is forking utilised to facilitate OSS development?

This research serves as a first step towards improving the understanding of forking in OSS and hopefully fuelling research in this direction for the benefit of OSS development by online communities. This paper is organised as follows. A review of the literature is given in Section 2. In Section 3, we describe our proposed conceptual model for social forking as a way to explain its key concepts. Section 4 presents our research methodology to empirically examine a web site that facilitates forking and OSS development, using our conceptual model as a guide. Section 5 provides our results and Section 6 presents our conclusions and a discussion of future work.

2 Literature Review

Forking is the creation of a new software project from the same code base as another [4] by anyone, with or without the knowledge or consent of the creator of the original project, as unlike commercial software licences OSS grants the right for anyone to access, modify and redistribute source code [14]. The definitions of forking vary subtly in emphasis. It can be merely seen as the splitting up of an OSS project into two or more projects which are then developed separately. [17]. Sometimes forking is weighed in with dire consequences. It is regarded as the splitting up of an OSS project into competitive [2] and incompatible [5] strands of OSS. Nevertheless, forking fosters the code base of an existing OSS to be moved in a different direction than that of its erstwhile project leadership [4].

It has been claimed there is a strong taboo against forking [5]. It is believed that forking divides an OSS community by weakening both user and developer involvement [11]. Projects forked from another dilute the attention of end users of OSS, the user base of the original project and the support network around it. They also starve the original project of developers, who need to split their effort for different forked projects [11]. Forks are also seen as a band aid solution for technical disagreements and interpersonal conflicts that cannot be resolved in the forking project [6]. Thus, leaders within an OSS community strive to directly resolve those issues within the forking project so as to reduce the need for forks [11].

Forking is seen to distance developers of an OSS community from the original project from which forks were created (i.e. the forking project) and make them lose interest in the project [11]. There is also the potential for duplication of effort in different forks, and rivalry among developers (e.g. claiming their forks are superior) [3]. A developer might face the loss of their reputation since (s)he is unlikely to contribute to multiple forks as well as the forking project and claim the credit for all his/her effort [5, 15]. Even when a developer can make modifications in a fork for a good cause, (s)he can feel a sense of rejection when modifications are but declined(?) by administrators of the forking project [7].

Forking has been touted as a danger to OSS development and its adoption [12] since forking has the potential for fragmenting the design into competing [2] and incompatible versions [5]. Whilst it is easy to create forks, the number of forks for a project, and hence variants of it, can explode, leading to a loss of commonality [7], the original intent and main characteristics of the software produced from the forking project.

On the positive side, forking permits specialisation of OSS for different needs [10]. For example, parties in an OSS community may use forking to extend or customise a standard implemented in the OSS to their advantage [19]. The OSS in a forked project can also evolve separately from the project from which it was forked to satisfy new requirements [4]. New features are experimentally developed in forks independently of the OSS from the original project. Variants of the OSS (e.g. for Apple iPhone vs. for Android) from the original project can also be developed in a fork. Hence, forking is also regarded as a source of innovation [2].

In an OSS development environment, an individual has some freedom to fork a project and choose to work on whatever part of the OSS suits his/her interest, agendas and approaches [11, 16], irrespective of time and geographical constraints. Forking also provides developers leeway in exploring alternatives for OSS [16]. Developers can also work at a fast pace without being bogged down by the bureaucracy of consensus-driven processes that manage changes [10]. When developers work on different forks, they have the opportunity to compete with one another by developing the OSS solution of best interest to their OSS community.

3 A Conceptual Model for Social Forking

To facilitate our investigation, we firstly define a conceptual model for social forking and its terminology. Forking is the task of creating a new software project from existing artefacts of another software project. Artefacts are not limited to source code. They can be anything related to the software that a project aims to develop, such as documentation, examples, test harnesses and third party libraries. The forked project (or simply the fork) can be used for enhancement, bug fixes, innovation and so on. A fork and the one from which it is forked forms a *successor-predecessor* relationship. A developer of a fork has a vested interest in the original project from which it was forked but the owner of the original project may not necessarily be aware of the forks created and their development activities.

When changes are made to the original project, the underlying DRCS notifies coders of its forked projects about the changes. They may then review the changes and incorporate them as an *update* to their forks as at their own pace, without any involvement of the owner of the original project. The social aspect of forking comes into play when social interaction occurs in order for a successor fork to make *contributions* [3] to a predecessor fork. In a simple case, a coder who has forked a project makes changes to it, notifies the owner of the predecessor project of the changes and interacts with the owner by exchanging comments about the changes. The owner then incorporates the changes into the predecessor project.

Forking is utilised at two layers of abstraction: endogenous and exogenous. *Endogenous forking* occurs within a community of social developers who work on forks for the same software product line. For instance, one creates a fork to enhance an existing feature of a software product. A fork tree for forks in a community can be represented as a tree structure. At the top of the tree is the *master* fork of the community from which all forks are created. *Primary* forks are those directly created from the master fork. Likewise, *secondary* forks are those created for primary forks.

Exogenous forking refers to the creation of forks across communities of social developers. Through exogenous forking, import and export relationships are established across community borders. In the former, a copy of a master fork in one community is *imported* into a master fork in another community. It can be used for bringing a copy of a third-party library into another project for creating a new software product. This import operation decouples activities of coders in one community from the other. The coders in the new community work with a baseline version or snapshot of the master fork, while those in the original community continue to evolve its master fork. Note also that the import operation only brings in whatever artefacts are required for the importing community. A more complex form of exogenous forking is to import forks from multiple communities into another community.

In an export relationship, artefacts in community B are purported to be an add-on or plug-in feature to artefacts in community A. Artefacts in B are essentially decoupled from those in B; the runtime code produced from artefacts in B can be run independently of those produced from A and vice versa. This independence also distinguishes between export and import relationships.

A fork is to a fork tree what a branch is to a version tree for version control systems in which branches are created for auxiliary tasks to be carried out [18]. Auxiliary tasks include performing fixes, distributed development, custom modification, dealing with conflicting updates [18]. Although both a fork tree and a version tree (as well as their artefacts) are structured similarly and managed under version control, a fork tree is differentiated from a version tree in a few ways:

- A fork can be used to initiate a separate strand of independent development for a new OSS. Unlike those in a branch, changes made in the fork need not be promoted to its predecessor;
- A fork can be created from only a *subset* of the artefacts from its predecessor whereas in a branching operation, a whole copy of a project's artefacts is made into its branch; and

- A fork can be created from *two* or *more* predecessors, thereby bringing features from different predecessors. A branch can only have one predecessor.

4 Research Methodology

A three-step empirical study was carried out to examine the phenomenon of social forking using our conceptual model as an analysis framework. In step 1, we chose and examined Github (https://github.com/) since it is one of the top websites based on the number of OSS development projects hosted[1]. We started our search for communities (i.e. software repositories in Github) using JavaScript programming language since JavaScript was the most popular in GitHub (20%) and it might also increase our chance of finding exogenous forking. We then narrowed our selection to those with the highest number of forks which was indicative of a high level of social activities and a rich source of forking data. In step 2, we developed and ran a tool against each of the software repositories identified to extract data from them via GitHub's public APIs (which are RESTful Web Services); to transform extracted data into a format based on our conceptual model; and to load the transformed data into a relational database. In step 3, the empirical data loaded in the database were explored to identify patterns of social forking. Our analysis of the results is presented next.

5 Results

The top nine JavaScript development communities hosted by Github having the highest numbers of forks were selected and used in our empirical analysis in November 2011. We investigated the relationships of the communities by examining their documentation and the files in their repositories. The communities exhibit import and export relationships and there are two variants in the former. In one case a full copy of the artefacts from community A is imported into community B whereas in the other case only a subset of the artefacts from A were imported into B. **Fig. 1** depicts the analysed communities and their relationships. The number labelled alongside each relationship instance represents the number of integration contributions made to a community as a result of maintaining the import/export relationships. This is further elaborated, together with the analysis of results for contributions, in Section 5.1.

5.1 Forks

All the communities we investigated had forks created at the primary and secondary levels, with two having forks at the tertiary level. Of the (7789) primary forks examined, 3.2% of them were used to create secondary forks and about the same ratio were used for the secondary forks. These figures are indicative that sub-communities were formed within the communities, which means developers participated in developing

[1] http://en.wikipedia.org/wiki/Comparison_of_open_source_software_hosting_facilities

features in the primary forks through their secondary forks. We observed that some primary forks were created to develop brand new innovations in the community based on their respective master forks, but were not necessarily intended to be incorporated into their masters. For instance, "nodejsjp/node" is a primary fork for "joyent/node" for the purpose of developing a Japanese version of "joyent/node". Indeed, these forks in conjunction with their successor forks formed sub-communities.

Of all the primary forks created, only 14% offered contributions to their respective master forks (11% for secondary to primary forks). A possible explanation for the high ratio of non-contributing forks (86% for primary and 89% for secondary) is that their developers were using forks to learn or experiment with the OSS's features only.

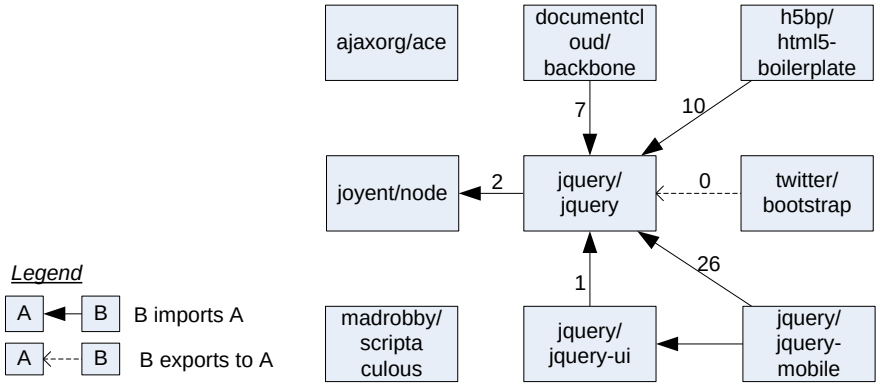

Fig. 1. Reviewed communities and their relationships

5.2 Contributions

The kinds of contributions made to each fork in the fork tree can be seen as a reflection of the ways in which their successor forks were used in each community. We analysed the titles and descriptions of over two thousand contributions and identified keywords which we subsequently used to categorise the contributions as defect fixes(43%), code enhancement(12%), documentation(7%), integration(2%), example enhancement(2%) and changes to test code and documentation(1%). The rest could not be categorised because of insufficient information. The integration category is triggered by the import relationship between two communities. For instance, when new features are added to artefacts in one community, they may also require import to the community that imported the original version of these artefacts. The artefacts in the latter community may also require changes to its artefacts in order to utilise the new features imported from the former community.

Of the contributions analysed from over one thousand contributing forks, the contribution rates (i.e. number of contributions divided by number of contributing forks) were 2.3 and 2.5 for contributing primary and secondary forks respectively. Note that contributions have the potential to be propagated further up the fork tree. This was evident from one case found during our examination of the contribution logs. A sec-

ondary fork ("flavaflav/jquery-mobile") produced a contribution to its primary fork ("arsduo/jquery-mobile") and the changes associated with that contribution were packaged as yet another contribution which was subsequently incorporated into the master fork ("jquery/jquery-mobile").

5.3 Users

Close to seven thousand developers created about eight thousand forks in the OSS communities analysed. We discovered some interesting behaviour of forking. In an extreme case one developer created forks in eight out of the nine communities analysed (i.e. working on projects in multi-communities). Six developers individually created more than one fork in the same community. This could be because each of these developers was using several forks to work on different parts of an OSS. 16% of developers made contributions to their OSS communities and from which we also identified some interesting behaviour of contribution. 5% of these developers made contributions to more than one community, with the top eight having made contributions to three communities. The most active developer made one hundred and forty-three contributions! The OSS community most attracted to contributions was jquery/jquery in which each developer created 3.3 contributions on average.

6 Conclusions, Limitations and Future Work

There has been a lack of critical studies into forking in OSS. In the literature there is a debate about whether or not forking should be used; some contend it as damaging to OSS development whilst some advocate its benefits to OSS communities and the OSS developed. To improve on this lack of knowledge about forking, we empirically investigated nine OSS communities from GitHub to gain an initial understanding of how it was used to facilitate OSS development in a social setting. It shows that forking was actively used by community participants for tasks such as fixing defects and creating innovations. "Forks of forks" were also utilised to form sub-communities within which specific aspects of an OSS product line were nurtured.

Since our study is in its infancy, we limited our analysis to one OSS hosting website, one programming language and nine communities, which poses validity threats to our findings. We thus plan to expand our study with additional web sites hosting OSS development (e.g. SourceForge, BitBucket, Gitorious) and interviews with OSS communities' members to gain better understanding of their perspectives on social forking. The excitement around social forking combined with our research to date will hopefully invigorate future research in this area and increase its uptake in practice.

7 References

1. Barcellini F, DÈtienne F, Burkhardt J-M, Sack W (2008) A socio-cognitive analysis of online design discussions in an open source software community. Interacting with Computers 20(1):141-165
2. Bitzer J, Schröder PJH (2006) The impact of entry and competition by open source software on innovation activity. In: J. Bitzer aPS (ed.): The Economics of Open Source Software Development 219-246
3. Cheliotis G (2009) From open source to open content: Organization, licensing and decision processes in open cultural production. Decision Support Systems 47(3):229-244
4. Ernst NA, Easterbrook SM, Mylopoulos J (2010) Code forking in open-source software: a requirements perspective. CoRR abs/1004.2889
5. Feller J, Fitzgerald B (2000) A framework analysis of the open source software development paradigm. Proc. 21st Int. Conf. Info. Systems 58-69
6. Fogel K (2010) Producing open source software. http://producingoss.com/en/index.html. Accessed on 15 Nov 2011.
7. Glass RL (2003) A sociopolitical look at open source. Communications of the ACM 46(11):21-23
8. Hertel G, Niedner S, Herrmann S (2003) Motivation of software developers in open source projects: an Internet-based survey of contributors to the Linux kernel. Research Policy 32(7):1159-1177
9. Lanubile F, Ebert C, Prikladnicki R, Vizcaíno A (2010) Collaboration tools for global software engineering. IEEE Software 27(2):52-55
10. Moen R (1999) Fear of Forking. http://linuxmafia.com/faq/Licensing_and_Law/forking.html. Accessed on 22 May 2011.
11. Muffatto M, Faldani M (2003) Open source as a complex adaptive system. Emergence 5(3):83-100
12. Nagy D, Yassin AM, Bhattacherjee A (2010) Organizational adoption of open source software: barriers and remedies. Communications of the ACM 53(3):148-151
13. O'Sullivan B (2009) Making sense of revision-control systems. Communications of the ACM 52(9):56-62
14. Open Source Initiative The Open Source Definition (Annotated). http://www.opensource.org/osd.html. Accessed on 01 June 2011.
15. Raymond E (1998) Homesteading the noosphere. First Monday 3(10)
16. Raymond E (1999) The cathedral and the bazaar. Knowledge, Technology & Policy 12(3):23-49
17. Stewart KJ, Gosain S (2006) The impact of ideology on effectiveness in open source software development teams. MIS Quarterly 30:291-314
18. Tichy WF (1985) RCS - a system for version control. Software: Practice and Experience 15(7):637-654
19. Vetter GR (2007) Open source licensing and scattering opportunism in software standards. Boston College Law Review 48(1)
20. von Krogh G, Spaeth S (2007) The open source software phenomenon: Characteristics that promote research. The Journal of Strategic Information Systems 16(3):236-253