Forking the commons: Developmental tensions and evolutionary patterns in open source software

Mehmet Gençer and Bülent Özel

İstanbul Bilgi University, Turkey
{mgencer,bulent}@cs.bilgi.edu.tr

Abstract. Open source software (OSS) presents opportunities and challenges for developers to exploit its commons based licensing regime by creating specializations of a software technology to address plurality of goals and priorities. By 'forking' a new branch of development separate from the main project, development diverges into a path in order to relieve tensions related to specialization, which later encounters new tensions. In this study, we first classify forces and patterns within this divergence process. Such tensions may stem from a variety of sources including internal power conflicts, emergence of new environmental niches such as demand for specialized uses of same software, or differences along stability vs. development speed trade-off. We then present an evolutionary model which combines divergence options available to resolve tensions, and how further tensions emerge. In developing this model we attempt to define open software evolution at the level of systems of software, rather than at individual software project level.

Keywords: Forking; Divergence; Specialization; Software Evolution.

1 Introduction

Beginning with its popularity as a commercially viable form of software innovation, open source development model has been often praised for its suitability for evolution and adaptation to fast moving demands on software products. On the other hand, understanding of software evolution in OSS research and practice remains to be confined to its closed source counterpart. This traditional conceptualization, in turn, uses the term software evolution as a synonym for software maintenance [8]. It acknowledges the environmental pressures on a single piece of software, and primarily concerns unpredicted changes in software through its life cycle.

This conceptualization is inadequate for systems of open source software. Unlike closed source software, open source software packages are forked or combined in a variety of ways. As such, environmental pressures and evolutionary processes work through systems of software, rather than a single software. In the face of openness, one needs a higher level unit of analysis to understand software evolution.

In this paper, we develop a theoretical model for evolution of systems of open software. We limit ourselves to cases of forking, re-forking, and occasionally, merging of forked variants. In our model we identify environmental or internal tensions on an OSS project, and patterns of consequent forking. Such forking may create separate species which no longer may exchange -genetic- code with one another (e.g. when fork uses a different, incompatible license, or is a result of power conflict), or may be a variant which can share code with its parent or sibling species (e.g. when fork is caused by stability/feature-richness trade-off).

Within this scope, we attempt to map essential elements of evolutionary framework to software. We suggest that through such models a better understanding of software evolution within the contextual dynamics of broader software ecosystem is possible, and can contribute to improve management and resource allocation in a variety of cases where OSS model is employed.

In this paper, we present our mapping of evolutionary elements to software, in the backdrop of existing literature. Summarizing empirical findings about forking patterns in OSS, we propose a model of evolutionary processes and dynamics around forking.

2 Software evolution vs. evolution in software ecosystems

Darwinian framework for biological evolution have been employed in explaining a variety of non-biological phenomena, primarily in economics. In doing so one needs to map the principal processes of variation, selection and inheritance. There is no random mutation in such social and economic systems but instead there are rational actions of human (or organization) actors. Thus the overall evolutionary analogy may be contested on the ground that variations are purposeful unlike those in biology. However, rationality in such complex systems is limited to information available to actors' to predict outcomes of their actions [6]. Complexity of outcomes in such systems make Darwinism particularly relevant to understand them [7]. Such an evolutionary framework has been used to explain economic and organizational systems [1].

In the field of software, the evolution concept has been used primarily through variations of Lehman's original conception [8], and almost interchangeably with the term 'software maintenance' [5]. Such usage of evolutionary framework, although weak, may be appropriate for proprietery software. On the other hand, in the case of OSS, life cycles of software projects exhibit complex patterns in which software packages are forked, merged, split, or combined in a variety of ways, thanks to their permissive copyleft or copycenter licenses. Apart from case studies on genealogy of certain OSS projects [5], however, attempts to analyze evolution above the unit of single software projects are rare.

On the other hand open source software seems to be particularly suitable for employing evolutionary framework. For any piece of software, creation and employment of its copies can be considered as corresponding to replication in biological evolution. What is different in OSS is the fact that many users modify it to fit their particular needs, thus mutating software. Depending on how

common such a need is, some modifications find their way into the main development branch of the software project. Such changes are replicated thereafter, hence becoming part of the species' gene pool. Certain others may correspond to unique needs, and may never leave the single site they are created. More interestingly there may be a variant which is demanded by a considerable user base, but it may not be possible to converge the mutated software with the main development stream for a variety of reasons (such as licensing, stability, target platform, feature incompatibilities, or power conflicts with leadership). Such are cases which correspond to creation of new species.

The brief articulation above lays out the parts of Darwinian analogy corresponding to variation and inheritance processes. With the selection process, the situation is even more similar to biological evolution. In OSS projects, even when corporate actors are involved [3], a species' access to resources in the environment corresponds to user and developer interest attracted to an OSS project. An OSS software project develops and becomes more appealing to a larger user base as developers prefer to contribute to it (rather than another software), unlike a proprietary software whose development may depend on corporate investment. Such developer support may depend on a variety of factors including appeal of design choices by initiators. However, the major factor is how the functionality provided by a new software corresponds to a niche need in the ecosystem, and how it compares to alternatives. Given the complexity of such an ecosystem, it seems plausible to assume that such correspondence (i.e. fitness in biological evolution) is largely unpredictable.

3 The open source and forking patterns

Since open source software is based on a commons based property regime, anyone can forgo to modify such a software technology for a special need. One way to to do this is to extend software capabilities in desired direction. In this process, which is called 'forking' in the open source community, a developer/group/firm starts (forks) a branch of development work separate from the rest of collaborators (the main branch). Such a fork faces an inherent paradox: (1) one may disregard what is going on in the main branch entirely, thus reducing constraints in terms of developing a capability, or (2) try to modify as few modules as possible to achieve the desired capability, using the rest of the modules from the main branch. The latter method keeps immediate constraints but allows one to continue using –hopefully useful!– collaborative development of the main branch. In many situations, one cannot evaluate and choose a subset of constraints beforehand (at least not easily), hence facing a choice between staying interdependent with others or going independent, with little or no shades of gray in between.

Current state of OSS licenses adds further complication to the matter. In contrast to a commercial license which was used to keep software innovation within a proprietary sphere of a firm, open source licenses were designed to keep them in public space. Thus first commercial firms who were interested in adaptability and innovation advantages of OSS were faced with a dilemma between

the power of collaborative innovation on the one hand and keeping competitive advantage on the other. Industry's answer to the problem was creating a variety of hybrid licenses (ie. copycenter licenses). While solving a range of competitive positioning problems, however, this introduced a new problem due to incompatibility of licenses preventing code sharing among projects in many cases [2]. Thus license incompatibilities enters OSS forking process as a potential complication.

In summary, independence and legitimacy 'to fork' under open source licensing regimes accommodates innovation and agility because it allows diversification to address tensions due to conflicting demands on development. It provides an assurance for each collaborator that they can go their way when there is a conflict of development goals.

In a previous study [4], we have observed various strategies based on forking, in response to a variety of tensions. We have found two broad categories. First one, *interdependent forks*, are the cases where the forked branch stays compatible with the parent branch. Such forks were triggered by needs of further specialization, differences in terms of stability/agility choices, etc. Further forks of the forks was possible, each with varying degrees of compatibility and mutual empowerment with other siblings. There were even cases of merging after a certain period of separation. The second category, *independent forks*, included cases where the fork became independent of the main branch. These were triggered by power conflicts, license issues, etc. In most cases in the latter category, only one of the branches survived.

New cases of forking has appeared since that study, some of which are more public than others. Among those are, for example, the Android system for mobile phones. Android forks the Linux kernel due to demanding requirements of mobile platforms, such as power consumption and user interface. In its current standing, the project have difficulties maintaining common code with the main branch, which introduces the danger of many vendors maintaining multiple versions of their hardware drivers for two different systems. Another example was the windowing system for Unix variants. Once dominant windowing system of XFree86 have changed its licensing scheme. The new license were incompatible with the copyleft licenses of many other software components in the Unix software ecosystem, of which it was a part. As a result the OSS community has created a fork named X Org, which soon became the dominant variant as the community abandoned the former one.

These observed cases of software divergence through forking can be classified as follows:

Variation - The fork creates two software variants which remain more or less compatible with one another. In effect, they become variations within the same species which retain advantage of code reuse or sharing. There are two major groups in this category: (1) Forks due to *specialization tensions*: An example is NetBSD fork of BSD Unix operating system. The fork was created to serve as a specialized variant which provides features for networking and security. The forked variants shared a large code base and kept empowering one another. (2) Forks due to *stability/agility tensions*: An example is Debian/Ubuntu Linux fork.

Ubuntu Linux was created to satisfy demands for using a feature rich Linux on the desktop systems, where Debian's focus was on stability and reliability. The two projects shared a large set of utility programs as well as benefiting from each others software package repositories.
Speciation -The fork creates two software species which are incompatible with one another, or effectively unable to share code. There are two major groups in this category: (1) Forks due to *licensing tensions*: An example is XFree86/XOrg fork. The fork was created when XFree86 project has adopted a licensing scheme which created a compatibility tension in the Unix ecosystem. The XOrg fork was created due to this tension, which eventually replaced the former. (2) Forks due to *power conflicts* within the leadership: An example is Emacs/Lucid Emacs fork. Lucid, a private company, has forked Emacs editor, triggering a series of power conflicts and trust issues with the original project's team. The two projects were not successful in aligning their efforts, hence went on their own way.

4 A model of divergence

Each fork, whether interdependent or independent, results from a tension. In time it ignites a new round of tensions. Several patterns are suggested by our previous study [3]. For example, in the case of GCC/EGCS fork, the fork was created due to differences in terms of stability and flexibility. While the fork served its purpose, the user community demanded the two projects to merge, hence creating a new tension. In contrast, the case of Debian/Ubuntu fork faced a different tension from its user base which valued usability promises of the Ubuntu fork over backwards compatibility with its parent.

In each of the cases (except the forks due to personal power conflicts), a fork, the consequent co-existence of two branches, and possible future mergers, seem to encounter tensions related to conflicting demands of specialization (flexibility, innovation speed, etc.) on the one hand and demands of compatibility (stability, collaborative efficiency, etc.) on the other. Our model, visualized in Figure 1, frames these observed patterns in a unified process.

The model visualization roughly corresponds to a timeline of events. An existing software community evaluates tensions regarding specialization, and a decision emerges about whether to fork, and if so whether in an interdependent or independent manner. In either case, but particularly interesting for us in the case of a fork, the -forked- project will continue for a while, with tensions are now relatively relaxed.

Survival of a forked branch faces several challenges such as being able to generate or sustain quality, keeping up attention of commercial or non-commercial users. If the targeted specialization corresponds to a growing niche and delivers the expected quality, it is likely that the project will survive and grow (in terms of users, and in turn in terms of developer resources contributing to it). Such a growth is likely to create new tensions of specialization. Depending on how the parent project is growing, it may also face demands to merge with its parent as well, since such a move will create certain advantages. However, if the fork was

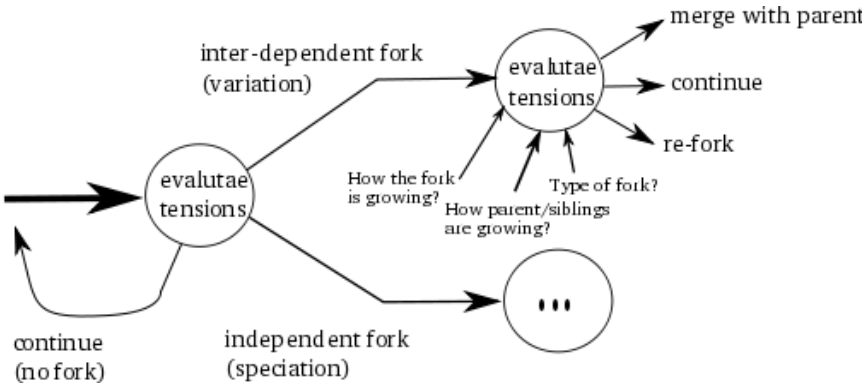

Fig. 1. A model of open software divergence in the face of tensions between plural interests/goals

an independent one (due to power conflicts, licensing differences, etc.) a merge is less likely even if it is desirable.

In summary, each phase of evaluating tensions of specialization and its result is effected by three factors: (1) how the fork is growing?, (2) how the parent or siblings are growing, and (3) the type of fork. The growth of fork itself possibly creates internal tensions for further specialization. The state of parent or sibling projects on the other hand causes developers to weigh advantages of staying separate versus advantages of merging with parent/siblings. Finally, the type of fork (interdependent/independent) further constrains the options to resolve tensions.

References

1. Aldrich, H.: Organizations Evolving. Sage (1999)
2. de Laat, P. B.: Copyright or copyleft?: An analysis of property regimes for software development. Research Policy, 34(10), 1511–1532 (2005).
3. Gencer, M., Oba, B.: Organising the digital commons: a case study on engagement strategies in open source. Technology Analysis & Strategic Management. 23(9), 969–982 (2011)
4. Gencer, M., Ozel, B., Tunalioglu, V. S., Oba, B.: Forking: The gpl coherent technology for flexible organizing in foss development. European Group of Organizational Studies Colloqium in Bergen, Norway (2006)
5. Godfrey, M., Tu, Q.: Evolution in open source software: a case study. In: Int. Conf. on Software Maintenance, pp. 131–142. (2000)
6. Hayek, F. A.: The use of knowledge in society. The American Economic Review. 35(4), 519–530 (1945)
7. Hodgson, G. M., Knudsen, T.: Why we need a generalized darwinism, and why generalized darwinism is not enough. Journal of Economic Behavior & Organization. 61(1), 1–19 (2006)
8. Lehman, M. M.: Programs, life cycles, and laws of software evolution. Proceedings of the IEEE. 68(9), 1060–1076 (1980)