Characterizing the OSS process

Andrea Capiluppi
Andrea.capiluppi@polito.it

Patricia Lago
Patricia.lago@polito.it

Maurizio Morisio
Morisio@polito.it

Dip. Automatica e Informatica
Politecnico di Torino
Italy

1. Introduction

The Open Source model of software development has gained the attention of both the business, the practitioners' and the research communities. The Open Source process has been described by the seminal paper by Eric Raymond [4] and [5]. However, sound empirical studies are still very limited [3], [6].

Our goal is to investigate the OS process by empirical means, to analyze, characterize it, and possibly model it with quantitative models. It should be noted that the Open Source process provides *open* process and product data, and therefore is a rare opportunity for empirical research.

Our initial research focus is on the characterization of the process, starting from the evolution of OS projects. In traditional projects, a significant number of releases in a short time is usually considered an instability factor [7] and [8], while in the OSS community, it is an evidence of vitality, shows the commitment of the authors and the power of attraction of other programmers [9]. Is it possible to characterize the vitality of projects? And, can vitality be traced to some other characteristics of a project?

2. The empirical study

We have considered two well-known Open Source portals (FreshMeat [1] and SourceForge [2]). They host around 10.000 projects. Using pseudo-random sampling we have selected a sample of 400 projects (mostly from FreshMeat). Each project is described by several variables (programming language, type of license, size of source code, type of documentation available and others). By indirect means (analysis of the Changelog file, or CVS) it is also possible to compute the number of people working on the project, and the number of external contributors. From FreshMeat we get both a vitality index, that considers the number of releases per time period, and a popularity index, which is a first measure of the interest of users to the project (project URL hits, mixed with subscriptions to it).

After several rounds of data exploration, both through automated techniques (clustering, correlation) and visual analysis of data, we have defined this tentative model.

- Independent variables: development mode (values: solo work vs. team work); patches (internal or external). Solo work means that a single person develops, team work that many develop. Internal patches means that patches and enhancements are made by the developers only, external means that patches and enhancements come form outside too.
- Dependent variables: vitality and popularity.

The two independent variables define four possible scenarios. We have observed that each of these four scenarios can be traced to a mix of vitality and popularity trend, and that, over time, a scenario can be followed by another, showing a kind of evolution path for OS projects. Further, each scenario points to typical values of other variables (such as documentation, license, etc). Let's analyze the four scenarios. An hypothesis about how a project could evolve from one scenario to another is sketched in the next section.

1- Solo work, internal patches. This kind of scenario is typically related to the first phases of an Open Source project: the main developer is doing most of the work, changes are enhancements or bug fixes. As it could be expected, the vast majority of projects fall into this scenario, as to testify that a lot of Open Source projects are in an early stage of evolution. To understand how long a project stayesin this first scenario, we evaluate its age through the count of the days during which it stayed on the portal. The code is generally small, there is no visible modularization (for example, through directories of files), but there is a scarce-to-adequate documentation. Vitality is generally low. On the other side, popularity has low-to-medium values. When a project can't get out of this scenario, it typically dries up, either for lack of work or for too high code complexity.

2- Solo work, external patches. Only one developer works on the project, but there is evidence of contribution from others for bug fixes or enhancements. Vitality is higher here.

As expected, the collaboration among programmers increases the number of releases per period, and popularity gains in response of it, giving more visibility to the project. The correlation between this scenario and the popularity attribute is quite clear in hti sscenario. Further there is usually a better documentation, which is frequently based on a complete user's manual and an increase of the code size, if compared to the projects referred in the first scenario; finally a better modularity of the source code.

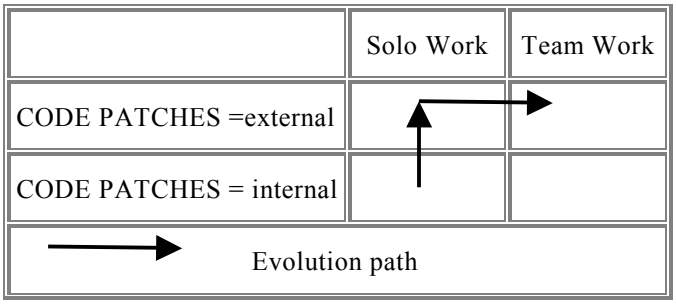
3-Team work, external patches. The third scenario is a further evolution of the two above, and is generally referred to as a 'mature' stadium: the main author still provides his work to the project, and the project itself would collapse without him, but the contributions of external programmers are not negligible. Compared with the case of a solo work, the team will get through more releases per period, which increases vitality: this in clearer expecially when the scope of the project, i.e. the application domain, is wide. The popularity is generally bigger than in the a solo work scenario. The presence of other users/developers is highlighted in the Changelog file, where some periodic entries are referred to other authors. The documentation is easily enhanced with internationalization, and often can grasp all of the internals, API's and howto's. The faster evolution is coupled with a higher status (tipically a stable/mature status) and an organization of the source code which reflects the actual division of work within the developers (one programmer is responsible for one directory). There are two general ways to get to this scenario: a project originated by a single user who accepted other developers, and a team of initial developers gaining enhancements by other programmers.

4-Team work, internal patches. In this scenario a well established team of developers develop and enhance their software, but they form a close community, without help from outside. A lower number of releases per period has the effect to decrease the vitality when compared to the scenario (3). No major differences are remarkable in terms of popularity. In this case, there are two different approaches to understand this scenario: the first one is a commercial firm that went Open Source to grab some of the advantages of this approach, but still not getting any concrete patches from its users. The second one is about a small-to-medium team working on a project and developing with a higher rate than a similar solo project, still not getting any patch. This second case gives an opportunity to understand the three levels of users defined in [3] : an Open Source software will dry up even in the case of a too small community using and giving any kind of feedback to the project.

3. Research questions

Our study has raised several interesting questions, that we will explore in the future.

1. How projects evolve from one of the four scenarios defined above to another? In its first stage, the project is developed by his author alone: feedback from others is initially low, but it is still useful to keep up its vitality. Feedback under the form of suggestions may become feedback under the form of patches and evolution to a teamwork mode. The second point of stability is related to the original author: when his work is not indispensable, an Open Source project is considered mature. We have not found any evidence to infere hypotheses about this second stage yet.



2. What is the relevance of the main developer in the evolution (growth, steady state or death) of an Open Source project?
3. What is the real role of the availability of documentation in the process of attracting developers in a project?
4. What is the role of a clear modularization of the source code?
5. Where is the limit (size, complexity, functions) of an Open Source project developed by a single programmer?
6. What is the role of the market size of a project (potential interested users and developers), and how can we characterize it?
7. How to define a 'success' factor for a project, are vitality and popularity enough?
8. What's the role of other project attributes such as license, programming languages and application domain?

4. References

[1] FreshMeat.net Web Page. On-line at http://freshmeat.net

[2] Sourceforge.net Web Page. On-line at http://sourceforge.net

[3] Mockus, A., Fielding, R. and Herbsleb, J. 2000. A case study of open source software development: the Apache server. *Proceedings of the 22nd International Conference on Software Engineering , May 2000, Limerick, IRL, pp 263-272.*

[4] Raymond, E., "The Cathedral and the Bazaar", *FirstMonday*, Vol.3 N.3 - March 2nd. 1998, on line at http://www.tuxedo.org/~esr/writings/cathedral-bazaar/

[5] Raymond, E., "Homesteading the Noosphere", FirstMonday, Vol. 3 N. 10 - October 5th. 1998.

[6] Stamelos, I., Angelis, L., Oikonomou, A., Bleris, G.L., 'Code Quality Analysis in Open-Source Software Development', *Information Systems Journal, 2nd Special Issue on Open Source Software*, 12(1), January 2002, pp. 43-60.

[7] Bezroukov, N., "Open Source Software Development as a Special Type of Academic Research", *FirstMonday*, Vol. 4 N. 10 - October 4th. 1999.

[8] Bezroukov, N., "A Second Look at the Cathedral and the Bazaar", *FirstMonday*, Vol. 4 N. 12 - December 6th 1999.

[9] Tirole J., Lerner J., "Some Simple Economics of Open Source", *Journal of Industrial Economics,* N. 52, July 2001.