Mining Version Histories to Verify the Learning Process of Legitimate Peripheral Participants

Shih-Kun Huang[1,2]
skhuang@csie.nctu.edu.tw

Kang-min Liu[1]
gugod@gugod.org

[1]Department of Computer Science and Information Engineering
National Chiao Tung University, Hsinchu, Taiwan
[2]Institute of Information Science, Academia Sinica, Taipei, Taiwan

ABSTRACT

Since code revisions reflect the extent of human involvement in the software development process, revision histories reveal the interactions and interfaces between developers and modules.

We therefore divide developers and modules into groups according to the revision histories of the open source software repository, for example, `sourceforge.net`. To describe the interactions in the open source development process, we use a representative model, Legitimate Peripheral Participation (LPP) [6], to divide developers into groups such as core and peripheral teams, based on the evolutionary process of learning behavior.

With the conventional module relationship, we divide modules into kernel and non-kernel types (such as UI). In the past, groups of developers and modules have been partitioned naturally with informal criteria. In this work, however, we propose a developer-module relationship model to analyze the grouping structures between developers and modules. Our results show some process cases of relative importance on the constructed graph of project development. The graph reveals certain subtle relationships in the interactions between core and non-core team developers, and the interfaces between kernel and non-kernel modules.

Keywords: Legitimate Peripheral Participants(LPP), Open Boundary, Open Source Software Development Process.

1. INTRODUCTION

Because of the success of Linux, GNU, Apache, and tens of thousands of open source development (OSD) projects in `sourceforge.net`, we review the process of OSD and compare it with conventional approaches to proprietary software development. Many researchers have explored and tried to explain the differences between the software processes of OSD and conventional approaches. Among them, Eric S. Raymond was the first to publish his findings in the noted *Cathedral and Bazaar* [11] model.

Ye and Kishida also proposed an open source software(OSS) development process model [15]. It is based on the evolving nature of a community with projects and a learning theory – *Legitimate Peripheral Participation (LPP)*, proposed by Lave and Wenger [6]. In [15], an OSS project may be associated with a virtual community, and developers may play certain roles in both the community and the project. During the learning process, the role of each member of the virtual community co-evolves in both the project and the community.

Few of the criteria of conventional software engineering methods, which are concerned with process models and control of schedules, can be applied in open source project development. In OSD, developers of a project may work together without knowing each other and build a successful system with millions of users worldwide. Although OSD does not appear to allow complete control and scheduling over software, it works well in reality. Besides, OSD projects often release new versions of software that are comparable to high quality proprietary software with similar functions. Such sustainable nature of the OSD process is worth exploring. However, although OSD has low initial deployment costs, there may be higher long-term costs.

In our experience, many open source developers do not contribute a great deal to OSD. They only do relatively minor work, such as fixing non-critical bugs, and do not make major contributions to the development process. Even so, although such minor contributors form weak links in developer networks, they are often a major driving force behind a project growing larger. This is similar to the small-world phenomenon.

The project-community evolutionary model, proposed by Ye and Kishida [15], states that any change of roles in the community maps to a change of roles in the project. The model also lists eight possible project roles and states that users, or peripheral developers, change their roles by learning about the project in detail, and are therefore central to the project.

In this paper, we propose a quantitative approach to analyzing the data of open source project development in order to evaluate the role changes of developers in a project. Through this analysis, we verify the learning process of LPP and provide a quantitative measurement for open source development models. The major advantage of our approach is that it is fully automatic; thus, manual verification is not required in the middle of the data mining process.

We believe that, in each open source project, there is a large amount of source code that does not need to be opened; that is, the success of an open source project depends on only a small proportion of its code. This would allow commercial developers of software to work with peripheral teams in the development of products without loosing control of their source codes.

2. METHODS

We use a similar approach to that of Luis et al [7]. For each target project, we perform network analysis of its version control repository. Our main source of data is `sourceforge.net`, which provides a full CVS repository archive.

From revision histories, we can construct social network graphs that represent the relations between developers of different parts of a project. The evolutionary pattern of a social network reflects some process features and anomalies during a project's evolution. With network analysis methods, we can measure the relative importance of each developer, and classify each one's role.

For each path, p, found in the revision log, we define a developer set, D_p, for path p (such paths refer to directories specified in the revision log.) Formally, we define a developer network graph as follows.

$$D_p = \{d | \text{developer } d \text{ has modified path } p\}.$$

Then, we can define a symmetric developer graph, G_d, as:

$$\begin{aligned} G_d &= \{V_d, E_d\} \\ V_d &= \{d | d \text{ is a developer}\} \\ E_d &= \{(d_1, d_2) | \exists \text{ path } p \text{ s.t } d_1 \in D_p \text{ and } d_2 \in D_p\}. \end{aligned}$$

In [7], the affiliation graph group is associated with the source code modules, while our group is associated with the directory. Our approach requires relatively less prior knowledge about the source code itself, and is more independent in terms of programming language; hence, it does not require human involvement to decide the affiliation group, as every step can be processed automatically.

We use the following definition in our analysis.

Distance Centrality (D_c) [13] : also called *closeness centrality.* The higher the value of D_c, the closer the vertices are to each other. Given a vertex, v, and a graph , G, D_c is defined as:

$$D_c(v) = \frac{1}{\sum_{t \in G} d_G(v,t)}. \qquad (1)$$

For each project, we first generate the developer social network, then compute the distance centrality of each node. From the distribution of the centrality values, we can discover the properties of different stages in the project development process.

3. RESULTS AND DISCUSSIONS

Figure 1 shows the developer social network for the project **awstats** [1]. Although this is a typical small project with only three developers, it has been very active according to `sourceforge.net`'s records. Its social network is fully connected, which means that all developers co-develop at least one directory. Project **phpmyadmin** [9] also has this kind of developer social network (Figure 2). In such a network, it is impossible to determine the importance of each developer, because they all have exactly the same attributes. Hence, we say that each developer plays the same role in the development process.

The above network pattern may reflect a possible flaw in our analytical method, because grouping developers based on directories is not detailed enough. However, it is also possible that the design of the software lacks proper modularity so that developers cannot modify a feature without modifying many directories in the source code.

The results of project **moodle** [2] (Figure 3) demonstrate another extreme case of social network patterns. A vertex's color represents its distance centrality value; the darker the color, the higher the centrality value. Nodes with the highest centrality values are rectangular in shape. The central portion of a node has only one vertex and all other vertices connect directly to that vertex. There are very few connections between non-central vertices. Project **filezilla** [5] (Figure 4) is another example of this kind of pattern.

Projects with this pattern start with a few developers deciding to work together, and they keep control of the source code as the project grows bigger. Non-central developers only make relatively minor contributions.

Nearly all projects with more then 10 developers have the same social network pattern as project **gallery** [8] (Figure 5). In such a pattern, only a small group of developers have a relatively high distance centrality, i.e., they are the center of the developer relationships; other developers play peripheral or intermediate roles. Project **bzflag** [12] (Figure 6) is another example of this kind of pattern.

Such social networks have many distance centrality values, which reflect many different kinds of project roles. Developers with high centrality values play important roles (core members or active developers), while those with lower values play peripheral roles (peripheral developers or bug fixers.)

Ye and Kishida [15] propose a project-community co-evolution process model, and define eight different roles in an open source project: **Project Leader**, **Core Member**, **Active Developer**, **Peripheral Developer**, **Bug Fixer**, **Bug Reporter**, **Reader**, and **Passive User**. Although, from the repository mining process, we are unable to associate each developer with a certain role, we can at least group developers into two large categories: active developer and

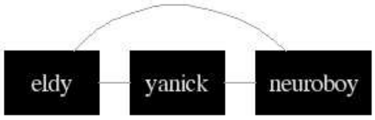

Figure 1: The awstats developer social network

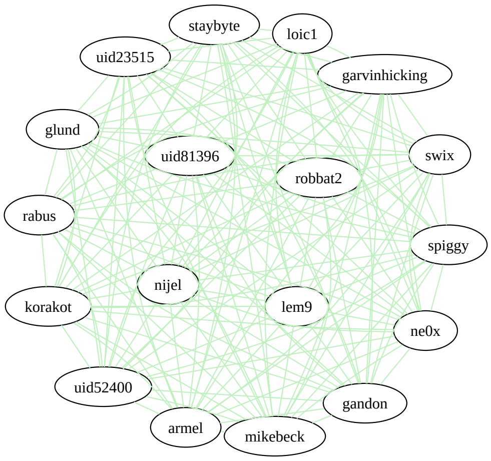

Figure 2: The phpmyadmin developer social network

above; and peripheral developer and below.

4. RELATED WORK

In 1999, Eric Steven Raymond proposed the community-based development model in his famous work *Cathedral and Bazaar* [11]. In this work, he takes the development process of the `fetchmail` project as an example and proposes the bazaar process development model.

Ye and Kishida [15], state that, in an open source project, "Every user is a potential developer," and propose a role hierarchy to show that participation in a project is actually a learning process for both peripheral users and core developers.

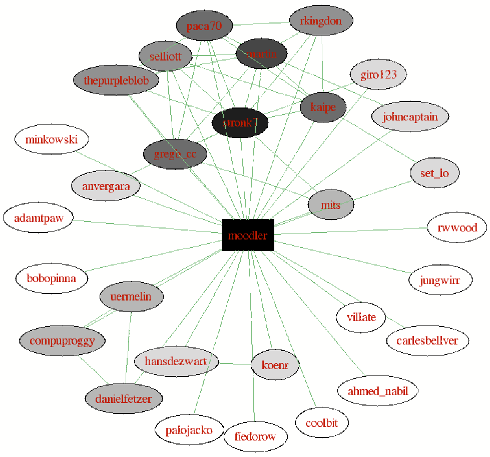

Figure 3: The moodle developer social network

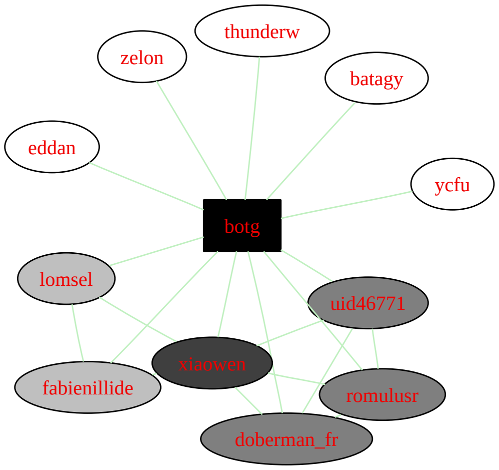

Figure 4: The filezilla developer social network

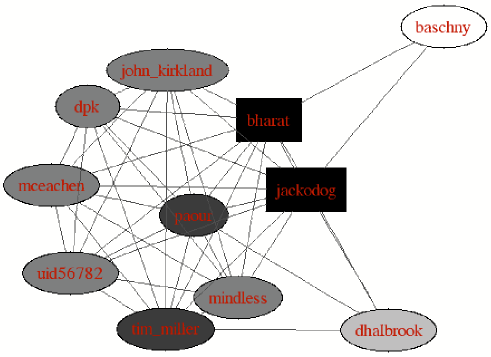

Figure 5: The gallery developer social network

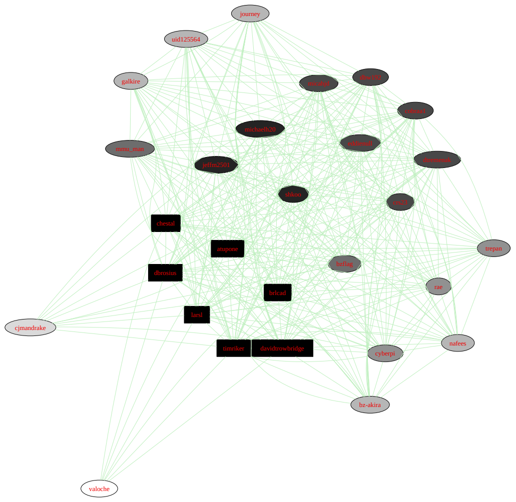

Figure 6: The bzflag developer social network

Project Bloof [10] gives a statistical revision log analysis for the source code evolution of a software project. The aim of Bloof is to help people comprehend software systems and the underlying development processes.

Project CVSMonitor [4] provides a more comprehensive presentation of revision analysis of the CVS repository, a version control system that has been widely used in the last ten years.

Zimmermann et al [16] recently proposed that mining version control histories can be helpful during the project development process, as they give programmers information about all the changes of a given revision.

White and Smyth [14] discuss several methods for analyzing large and complex network structures. In their experiments, they evaluated the different properties of many algorithms on toy graphs and demonstrated how their approach can be used to study the relative importance of nodes in real-world networks, including a network of interactions among the September 11th terrorists, a network of collaborative research in biotechnology among companies and universities, and a network of co-authorship relationships among computer science researchers.

Scacchi and Jensen [3] use techniques that exploit advances in artificial intelligence to discover the development processes of publicly available open source software development repositories. Their goal is to facilitate process discovery in ways that use less cumbersome empirical techniques and offer a more holistic, task-oriented process than current automated systems provide.

5. CONCLUSION

In this work, we use social network analysis methods to analyze the developer social network of a project created from the project's revision history.

We then try to verify the LPP process in Ye and Kishida's work [15]. Although this is not very accurate, we can at least split project developers into two groups: core and peripheral. This supports our conjecture that even in an open source project, there is a part of the source code that can be retained by core members only. With further graph-based network analysis, we believe that it would be possible to achieve more accurate results.

Developers involved in the revision process reveal their skill and familiarity with the source modules by different degrees of interfacing and interaction with core members. From the revision histories, we build a link structure between developers and code modules and analyze the relationships between these structures to determine their level of involvement with core teams and kernel modules. The extent of developers' involvement can be ranked. From the ranking results, we can verify the LPP learning process and propose a potential boundary between conceptual kernel and non-kernel modules. This boundary gives a clear indication of the degree of source code openness in joint development projects involving core and none-core teams of developers . The weak links around the boundary may significantly affect the ability of external peripherals to maintain the project's vitality and popularity. Our preliminary results reveal a few such process cases of relative importance on the constructed graphs that could affect a project's development.

6. REFERENCES

[1] Laurent Destailleur. Awstats is a free, powerful server logfile analyzer with many features, 2000. http://sourceforge.net/projects/awstats/.

[2] Martin Dougiamas and Eloy Lafuente. Moodle is php courseware that aims to make quality online courses (e.g., distance education) easy to develop and conduct., 2001. http://sourceforge.net/projects/moodle/.

[3] Chris Jensen and Walt Scacchi. Data mining for software process discovery in open source software development communities. In *Proc. Workshop on Mining Software Repositories*, page 96, 2004.

[4] Adam Kennedy. Project cvsmonitor. cvsmonitor is a cgi application that analyzes cvs repositories in a much more useful and productive way, 2002. http://ali.as/devel/cvsmonitor/.

[5] Tim Kosse. Filezilla is a fast ftp and sftp client for windows with many features; filezilla server is a reliable ftp server, 2001. http://sourceforge.net/projects/filezilla/.

[6] J. Lave and E. Wenger. *Situated Learning: Legitimate Peripheral Participation.* Cambridge University Press, Cambridge, 1991.

[7] Luis Lopez-Fernandez, Gergorio Robles, and Jesus M. Gonzales-Barahona. Applying social network analysis to the information in cvs repositories. In *MSR2004*, 2004.

[8] Bharat Mediratta and Chris Smith. Gallery: A slick, intuitive web-based photo gallery with authenticated users and privileged albums, 2000. http://sourceforge.net/projects/gallery/.

[9] Oliver Müller, Marc Delisle, and Loïc Chapeux. phpmyadmin is a tool written in php for handling the administration of mysql over the web, 2001. http://sourceforge.net/projects/phpmyadmin/.

[10] Lukasz Pekacki. *Project Bloof: an infrastructure for analytical processing of version control data.* 2003. http://sourceforge.net/projects/bloof/.

[11] Eric Steven Raymond. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary.* O'Reilly, 1999.

[12] Tim Riker and David Trowbridge. *OpenSource OpenGL Multiplayer Multiplatform Battle Zone Capture the Flag: a 3-D first person Tank Simulation.* 2000. http://sourceforge.net/projects/bzflag/.

[13] Gert Sabidussi. *The centrality index of a graph*, volume 31, pages 581–603. Psychometrika, 1966.

[14] Scott White and Padhraic Smyth. Algorithms for estimating relative importance in networks. In *KDD '03: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 266–275. ACM Press, 2003.

[15] Yunwen Ye and Kouichi Kishida. Toward an understanding of the motivation of open source software developers. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pages 419–429. IEEE Computer Society, 2003.

[16] T. Zimmermann, P. Weigerber, S. Diehl, and A. Zeller. Mining version histories to guide software changes. In *26th International Conference on Software Engineering (ICSE 2004.)*, 2004.