Evolution Patterns of Open-Source Software Systems and Communities

Kumiyo Nakakoji[1,2,3] Yasuhiro Yamamoto[2,4] Yoshiyuki Nishinaka[1] Kouichi Kishida[1] Yunwen Ye[1,5]

[1]SRA Key Technology Laboratory
3-12 Yotsuya, Shinjuku, Tokyo, 160-0004, Japan
+81-3-3357-9011

[2]Grad. School of Information Science, NAIST
8916-5, Takayama, Ikoma, Nara, 630-0101, Japan
+81-743-72-5381

[3]PRESTO, JST [4]Japan Society for the Promotion of Science [5] University of Colorado at Boulder

{kumiyo, yxy}@is.aist-nara.ac.jp {nisinaka, k2, ye}@sra.co.jp

ABSTRACT

Open-Source Software (OSS) development is regarded as a successful model of encouraging "natural product evolution". To understand how this "natural product evolution" happens, we have conducted a case study of four typical OSS projects. Unlike most previous studies on software evolution that focus on the evolution of the system per se, our study takes a broader perspective: It examines not only the evolution of OSS systems, but also the evolution of the associated OSS communities, as well as the relationship between the two types of evolution.

Through the case study, we have found that while collaborative development within a community is the essential characteristic of OSS, different collaboration models exist, and that the difference in collaboration model results in different evolution patterns of OSS systems and communities. To treat such differences systematically, we propose to classify OSS into three types: *Exploration-Oriented*, *Utility-Oriented*, and *Service-Oriented*. Such a classification can provide guidance on the creation and maintenance of sustainable OSS development and communities.

Keywords

Open-Source Software (OSS), software evolution, Open-Source Software community, case study

1. INTRODUCTION

Open-Source Software (OSS) refers to software systems that are free to use and whose source code is fully accessible to anyone who is interested. Most OSS systems start out with a developer who wants to solve his or her own particular problem and makes the solution (system) available to others for free. Because it is free, it often attracts many users who have a similar problem, and because of the free access of source code, some interested users become co-developers by extending or improving the initial system. Together with the original developer, users and co-developers create a collaborative OSS community around the system. Without such OSS communities, OSS projects are not likely to be successful.

Most OSS systems are not necessarily carefully designed in advance. They evolve in response to the needs of users in the OSS community, and the evolution is carried out by contributing (co-)developers of the same community. Although the evolution of an OSS system is not well planned, "giving users of a product access to its source code and the right to create derivative works allows them to help themselves, and encourages *natural product evolution* as well as preplanned product design [14]."

To understand how this "natural product evolution" happens in OSS systems, we have conducted a case study of four typical OSS projects. Unlike most previous studies [1, 7, 9] on software evolution that focus on the evolution of the system per se, such as the growth of size, the decay of architecture and design, and the change of defect density, our study takes a broader perspective: It examines not only the evolution of OSS systems, but also the evolution of the associated OSS communities, as well as the relationship between the two types of evolution.

Although an OSS project might have a leader (often the one who initiates the project), the leader neither has a grand plan for the system at the beginning, nor dictates the evolution of the system. It is the whole OSS community that collaboratively drives, as both users and developers, the evolution of the system. Therefore, a full understanding of the evolution of an OSS system cannot be complete without understanding the evolution of the OSS community and its role in driving the evolution of the system.

This paper reports the case study in which we analyze and compare four different OSS projects and communities that our company, SRA Inc., is involved. Through the case study, we have found that while collaborative development within a community is the essential characteristic of OSS, different collaboration models exist, and that the difference in collaboration model results in different evolution patterns of OSS systems and communities. To treat such differences systematically, we propose to classify OSS into three types: *Exploration-Oriented*, *Utility-Oriented*, and *Service-Oriented*. Such a classification can provide guidance on the creation and maintenance of sustainable OSS development and communities.

In what follows, we first briefly describe the background of the case study, followed by an overview of the four projects. We then compare the four projects and analyze their differences in collaboration model and evolution pattern of the system and community. Based on the analysis, we identify three types of OSS projects. The paper concludes with a discussion of how the identification of the three OSS types can help us better understand the evolution of OSS systems and communities.

2. BACKGROUND OF THE CASE STUDY

In January 2001, the Information technology Promotion Agency (IPA) of the Ministry of Economy, Trade, and Industry (METI) of Japan, decided to conduct a survey on the current status of OSS

development in the Japanese software industry. SRA was awarded the grant to conduct the survey, which identified different types of OSS projects, and compared existing industrial and governmental support for OSS development in different countries. SRA is a leading company in the open source movement, and has been supporting the activities of the Free Software Foundation (FSF) since 1987. In addition, SRA has carried out a variety of open source software development projects within its Open Source Business Division. This paper is based on a part of the findings from the survey, which is a case study over four different OSS development projects conducted within SRA.

The four projects we have studied are:

1. the *GNUWingnut* project, which provides support for a number of GNU software applications, such as GCC and Emacs, for Japanese companies that need GNU software ported into their hardware platforms;
2. the *Linux Support* project, which offers support for Linux users as a master System Integration distributor;
3. the *SRA-PostgreSQL* project, which supports Japanese customers who use the PostgreSQL database; and
4. the *Jun* project, which is a 3D graphic and multimedia library for Smalltalk and Java.

A survey was conducted by interviewing the project members of each of the four OSS projects. During the interview, we asked questions including:

- what open-source software they are dealing with;
- how the development of the open-source software has been done;
- what communication media the developers use in the development of the software;
- how they do business with the open-source software; and
- what benefit they see by doing business with open-source software.

Instead of obtaining simple answers, we ask those questions as a way of starting a series of discussions. In addition to the interviews, we examined their mailing-list archives and quantitative data related to particular aspects when necessary.

Note that the open-source software we describe in this paper reflects the views of those with whom we conducted the study. The views and opinions expressed by the project members who were interviewed may not be consistent with that of the core members of each project. For instance, we have interviewed the SRA-PostgreSQL project members at our company, but we have not interviewed with the PostgreSQL core development team members. This case study is to report how the OSS development project members at a for-profit company view their OSS development, and how different types of OSS development results in different types of business projects.

3. THE FOUR PROJECTS

This section describes the four projects in detail.

3.1 The GNUWingnut Project

As the name suggests, this project deals with GNU (http://www.gnu.org/) software developed by FSF (Free Software Foundation). The GNUWingnut project helps clients import GNU software programs onto their particular hardware platforms. GNU is a project that tries to develop a "free" UNIX operating system, organized by Richard Stallman at FSF. For Stallman and FSF, programs are "scientific knowledge to be shared among mankind" [6]. That is, for them, software is knowledge developed by "highly trained professional programmers" and therefore to be shared among human beings in the same way as the knowledge medical doctors develop is shared through research papers and books. It is this spirit that makes their software *free*. They have been using the term "free" not to mean that the software is free of charge but the source code is free to view, modify, and distribute under the license called GPL (GNU Public License) with the ownership notion called copy-left.

It is not our purpose here to describe GNU and FSF in detail, however, several interesting characteristics are worth to mention. Although it may not be explicitly stated, this view of programs as scientific knowledge has developed a culture where open-source systems need to be of very high quality; they want to develop the "correct" and "best" system for implementing a piece of functionality. Because it is to be good and to be shared among mankind, centralized control has been exercised. GNU software development teams observe strict coding rules and format guidelines [5] to make their software to be easily shared among mankind. Only one core version of the software is allowed and variations and alternatives need to be integrated within the core version. All bugs found need to be reported so that the core members can fix them. Overall, control is very much centralized.

The main task of the GNUWingnut project is to help clients port GNU programs into their target machines. A typical case is that a hardware vendor needs to have GNU Emacs and GCC to run on their super-computer operating systems. This involves two types of work. The first type of work is to develop patch programs for the clients. Although the source code is available, many GNU programs are very large and complex and require substantial knowledge and expertise to understand. The GNUWingnut project members offer such expertise, enabling clients to develop patch programs faster and better.

The second type of work, which is more interesting and possibly unique to GNU-related software development, is to help clients increase the quality of patches by revising and refining them so that they can be reported back to the GNU core development team. The clients need such help from the GNUWingnut project for the following three reasons.

First, as mentioned above, a GNU project wants to have a single version for a particular program and all bug fixes and updates need to be reported back in to the core development team. For instance, when a super-computer vendor develops a patch of GCC for their super-computer operating system, this company needs to have this patch reported back to the GCC core development team and to be incorporated into the core version; otherwise they have to develop a patch for every subsequent release of GCC.

Second, as also noted above, GNU requires fed-back programs strictly adhere to GNU guidelines for coding, formatting, and documenting. Although most of these guidelines can be enforced by using appropriate "modes" in Emacs, it still requires skills in observing these guidelines.

Third, a "cultural barrier" for Japanese programmers keeps them from directly communicating with the GNU core members through mailing-lists. Many programmers in Japan view the GNU core team as a group of super-programmers with highly respected skills, and want to keep a "respectful distance" from them. Some of the GNUWingnut project members have been closely working with the GNU core members for the last decade, and they serve as the intermediary between the clients and GNU core members.

3.2 The Linux Support Project

The Linux Support project at SRA provides user support for the Linux operating system, excluding the Linux kernel. We make this distinction because, similar to GNU, the Linux kernel development is under centralized control, while the remainder of Linux has been developed in the bazaar style [16] with decentralized control. The Linux support project supports the bazaar model, and accordingly in this paper when we refer to Linux we are referring to the portions of Linux outside the kernel unless specifically noted.

Contrary to the GNU programs, multiple versions of programs for the same functionality exist in Linux. No official centralized repositories have been developed for Linux OS peripheral tools, such as device and printer drivers. Each developer puts their programs on the Web; and users rely on Web search engines to find the needed program. Because multiple versions exist all over the world (i.e., the World Wide Web), directory services are necessary to find the needed programs. Furthermore, there are programs that are not compatible with each other. Distribution packages have been developed to help users find programs that are compatible with each other.

O'Reilly specifies four types of business models to deal with OSS [13]: (1) Support Seller, (2) Loss Leader, (3) Widget Frosting, and (4) Accessorizing. The Linux Support project at SRA can be characterized as Support Seller that helps customers identify and solve problems in the course of using Linux. A typical task is to help clients find appropriate distribution packages and to customize software for their needs. Linux Support Project members are also asked to find up-to-date information on security and bug reports related to their clients' Linux programs, which are scattered all over the Web.

Therefore, members of the Linux Support project are required to be able to find the needed information and to read and understand source code produced by others. For instance, if a bug is found in a Linux program, a typical process taken by a project member is as follows:

1. Read the newest version of the source code to see if the bug is fixed.
2. If not, read the released version of the source code to see if the bug is fixed.
3. If not, check a bug tracking report produced by the distributor if it reports the bug.
4. If not, check related mailing lists to see if the bug has been reported.
5. If not, try to find Web pages that report similar bugs.

When they find a newly-fixed program, they typically use the *diff* command to see how the bug is fixed, and apply the changes to the existing source code.

Surprisingly, project members develop patches for their customers and fix bugs, but they do not necessarily contribute the patches back to the community. The leader of the project explains that the customers do not care about version updates and they prefer to stay with the current version of the system as long as the system is working, even if new versions are available. This is very different from the GNUWingnut project in which it is critical that the patches developed and used at a customers' site get incorporated into the core version because otherwise they would be left behind. Once incorporated, their drivers and interfaces will be taken care of by the GNU core development team.

3.3 The SRA-PostgreSQL Project

The *SRA-PostgreSQL* project deals with the PostgreSQL database (http://www.PostgreSQL.org/), which is an open source system originally developed as a research prototype. The system has evolved and is now comparable to large-scale commercial database systems.

Because robustness is highly desired in database systems, PostgreSQL is strictly controlled by the core development team and the major development team. Decisions about the development of PostgreSQL are made democratically in the development team that communicate through mailing lists. Most discussions in the mailing lists are not concerned with the implementation and source code, but with specifications of new features and requirements, because for a database system, any change in specification may affect the overall performance and quality of the system.

The primary task of the SRA-PostgreSQL project has been internationalization. This has been done in four steps: first, the SRA-PostgreSQL project members have locally developed patches to deal with the Japanese language. Second, they have extended the patches so that they are able to deal with any two-byte code languages. Third, the patches have been incorporated in the core version of the PostgreSQL system. Finally, the now-internationalized PostgreSQL has become the standard distribution version.

Because of the contribution made by the SRA-PostgreSQL project team to the large PostgreSQL community, the leader of the SRA-PostgreSQL project has become a member of the major development team.

In addition to internationalization, the SRA-PostgreSQL project helps Japanese clients to port the system to multiple platforms, and conducts benchmark test. The project also serves as the Japan center of PostgreSQL, providing a Japanese ftp site for bug fixes and patches. Many Japanese users used to have trouble finding necessary information because most information regarding PostgreSQL is written in English. The SRA-PostgreSQL project helps them by translating English documents into Japanese.

For PostgreSQL, the biggest advantage of being open-source is that people can find bugs more quickly—bugs become shallow with enough eyeballs [16]. Most users in the PostgreSQL community contribute by testing and finding bugs through examining the source code, rather than writing code.

Another interesting aspect of the PostgreSQL project comes from the fact that the software is a database system. When reporting a bug, it is often necessary to use a specific set of data to reproduce the bug, because without the data it is very difficult for developers to understand what the problem is and to debug. However, such data is often proprietary and cannot be made public. Therefore, customers often ask the SRA-PostgreSQL project members to debug it, ensuring that their data are available only to the project members rather than the whole PostgreSQL community. The SRA-PostgreSQL project members then develop patches and send them to the PostgreSQL core development team.

3.4 The Jun Project

The *Jun* project at SRA develops and distributes the Jun library (http://www.sra.co.jp/people/aoki/jun/), a Smalltalk and Java library for 3D objects and multimedia data handling. Different from the above three projects, this project deals with the software that has been developed in-house. We have reported how the Jun

library has evolved over the last five years and how centralized decision making and continuous evolution has been achieved [1].

As reported in [1], not only the source code of Jun, but also its underlying object model has been used by the Jun community. Jun has served as a reference model in the development of 3D objects and multimedia data handling. Jun's evolution differs from other open-source systems. Instead of a wide community of programmers each contributing a small part, almost all of Jun is developed by a small group of three to five programmers. The development process is strictly controlled by the single project leader, who does both quality control and decision making in terms of which directions the project should evolve. Although the community does not contribute much source code, it provides feedback, feature requests and bug notices.

The business model of the Jun library is to develop software applications for customers with Jun. Although Jun is an open source library and is freely available for other development organizations and developers, the library has become quite large, creating a high learning curve for understanding and applying it. As its original developer, the Jun project team has an obvious advantage over other developers. The project members have been asked to develop research application systems using Jun, and to offer consultation on the use of Jun as well as the use of underlying models.

4. ANALYSIS

This section compares and analyzes the four projects, using a general framework we have developed for understanding OSS systems and communities.

4.1 The General Framework

As stressed in Section 1, to fully understand the nature of evolution in OSS projects, we have to examine both the system and the community, which is the driving force of the evolution. Therefore, our general framework for analyzing OSS projects is based on the roles that OSS community members play in the community, and the structure of the community defined by the collaborative relationship among those different roles.

4.1.1 Roles of Community Members

One distinct feature of an OSS project, as compared to the commercial software development, is that members of the OSS project assume certain roles by themselves according to their personal interest in the project, rather than being assigned a task by someone else. A member may have one of the following eight roles (Figure 1).

Passive User. Passive Users just use the system in the same way as most of us use commercial software; they are attracted to OSS mainly due to its high quality and the potential of being changed when needed.

Reader. Readers are active users of the system; they not only use the system, but also try to understand how the system works by reading the source code. Readers are like peer reviewers in traditional software development organizations.

Bug Reporter. Bug Reporters discover and report bugs; they do not fix the bugs themselves, and they may not read source code either. They assume the same role as testers of the traditional software development model.

Bug Fixer. Bug Fixers fix the bug that is either discovered by themselves or reported by Bug Reporters. Bug Fixers have to read and understand a small portion of the source code of the system where the bug occurs.

Peripheral Developer. Peripheral Developers contribute occasionally new functionality or features to the existing system. Their contribution is irregular, and the period of involvement is short and sporadic.

Active Developer. Active Developers regularly contribute new features and fix bugs; they are one of the major development forces of OSS systems.

Core Member. Core Members are responsible for guiding and coordinating the development of an OSS project. Core Members are those people who have been involved with the project for a relative long time and have made significant contributions to the development and evolution of the system. In some OSS communities, they are also called Maintainers.

Project Leader. Project Leader is often the person who has initiated the project. He or she is responsible for the vision and overall direction of the project.

Not all of the eight types of roles exist in all OSS communities, and the percentage of each type varies. Each OSS community may have different names for the above roles. For the sake of comparison, we will use above names throughout the paper.

4.1.2 Community Structure

Although a strict hierarchical structure does not exist in OSS communities, the structure of OSS communities is not completely flat. The influences that members have on the system and the community are different depending on what role they play. Figure 1 depicts the general layered structure of OSS communities, where the role closer to the center has a larger influence. In other words, a Project Leader has a larger influence than a Core Member, who in turn has a larger influence than an Active Developer, and so on. Passive Users have the least influence, but they still play an important role in the whole community. Although they do not directly contribute to the development of the system technically, their very existence contributes socially and psychologically by attracting and motivating other more active members, to whom a large population of users is the utmost reward and flattery of their hard work [16].

The roles and their associated influences in OSS communities are not associated with any attributes (such as age, title, etc.) of a member; instead, they can only be earned through contributions to the community. Roles are not fixed either; each member can play a larger role if they aspire and make appropriate contributions.

It is important to maintain a balanced composition of all the different roles in a community, otherwise an OSS community is not sustainable [12]. At one extreme, if all the members are Passive Users, the OSS system never evolves. At the other extreme, if all the members are Core Members, it is very difficult to coordinate all the development efforts and the further evolution of the system is also unsustainable.

Each OSS community has a unique structure depending on the nature of the system and its member population. The structure of an OSS community differs at the percentage of each role in the whole community. Generally speaking, most members are Passive Users. For example, about 99% of people who use Apache are Passive Users. The percentage drops sharply from Readers to Core Members. Most open source software is contributed only by a small number of developers [12, 14].

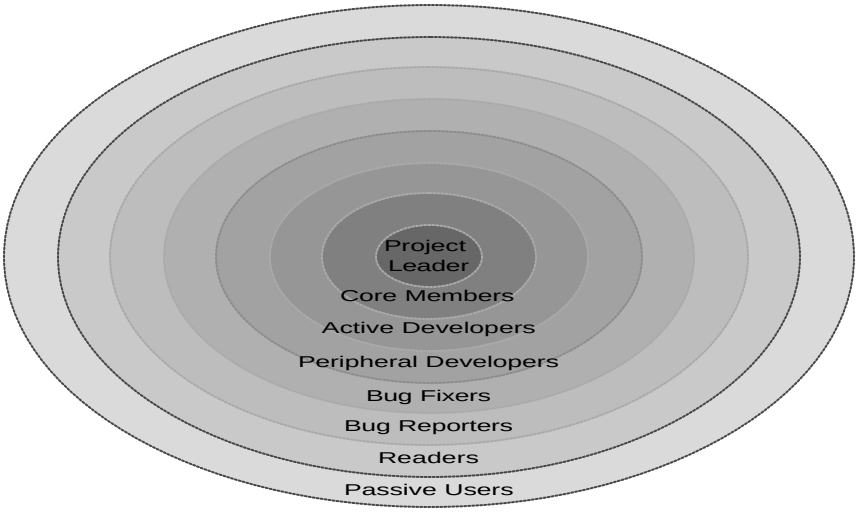

Figure 1: General Structure of an OSS Community

4.2 Roles and Structures of the Four OSS Communities

This section analyzes the roles and structures of the four OSS communities we have studied according to the general framework. Except for the Jun project that was initiated in SRA, all the other three projects are a portion of larger OSS communities, therefore, our analysis will be aligned with the whole large communities with special focus on the roles that SRA people play.

Most GNU systems, represented by GCC and Emacs, have a Project Leader who is often the person who initiates the system. In a few GNU systems, Project Leaders might be helped by several Core Members. The Project Leader makes most of the decisions about the development of the systems. Although all the other members are free to contribute and providing feedback, it is up to the Project Leader to decide which contribution should be included and which feedback should be addressed. Due to the limitation of available time, the Project Leader is not able to deal with all the contributions and feedback, many of which go unnoticed. Therefore, Active Developers, whose capability is well regarded and trusted by the Project Leader as well as the community, and whose number is not very large, not only contribute their own code but also play an intermediary role of (1) improving the code contributed by Peripheral Developers and Bug Fixers and (2) recommending the code to the Project Leader for their incorporation into the core version. Members of the GNUWingnut project are mainly Active Developers in the whole GNU community, and they mediate the communication from less recognized contributing members to the Project Leader.

Because GNU systems are meant to be "scientific knowledge to be shared among mankind" and they are developed by highly respected expert programmers, a large percentage of Readers exist in the GNU community. They learn programming skills and knowledge by reading the freely accessible source code. At the same time, they are also acting as peer reviewers to ensure the quality of the system.

The whole Linux community has one Project Leader who is Linus Torvalds, and has a few Core Members who are responsible for the development of subsystems. It also has many Active Developers who are working independently on subtasks, such as specific device drivers. However, there are far more Passive Users who do not care about the source code nor be able to understand the code. This breeds the need of supporting those Passive Users. Members of the Linux Support project at SRA are Peripheral Developers, Bug Fixers, and Readers. They understand the code and are able to find a fix for the bugs, or fix the bugs by themselves when the bugs are reported by customers who are Passive Users.

PostgreSQL does not have a single Project Leader. Instead, it has six Core Members who communicate with each other through a dedicated mailing list to discuss and decide the direction of the system. The inclusion of a new feature must be sponsored by one Core Member and voted by all Core Members. The community also has about 30 Active Developers (major development team). The programs developed by Active Developers exist as patches, and are finally incorporated into the core version only after they are approved by the Core Members. Most community members are Passive Users and Bug Reporters. Few Readers exist. The population of Bug Fixers is quite low too. The leader of the SRA-PostgreSQL project is an Active Developer, and the other project members work with the leader as Peripheral Developers.

The Jun community has a Project Leader who is an SRA employee. Several other SRA employees work together with him as Core Members. All the development is conducted in SRA. Because public version is released after rigorous internal tests, few Bug Reporters exist in the community. Most members are Passive Users. A few members are Readers who study Jun thoroughly and use it as a reference model to create a similar system in another language [1].

4.3 Evolution of Systems and Communities

After having defined the roles and structures of the four OSS communities according to the general framework, we are ready to analyze the evolution patterns of the systems and the communities, as well as their mutual dependence.

4.3.1 Evolution of the Systems

Because each of the four OSS projects has different objectives, their evolutionary patterns also differ. Figure 2 gives a schematic picture of how evolution takes place in each project.

GNU aims to have a single, clean, nice, well-written version of implementation for a single piece of functionality. Therefore,

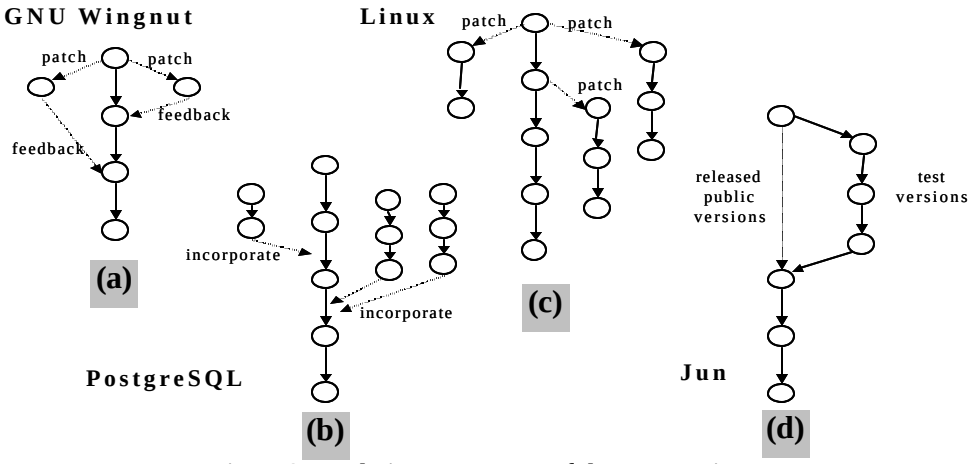

Figure 2: Evolutionary Patterns of the Four Projects

when other people develop their own patches, the patches are distributed only when they are incorporated into the core version.

In Linux, on the other hand, there is much less motivation and encouragement for contributing back the developed patches, as we have discussed in Section 3.2. Multiple implementations for the same functionality are allowed. Therefore, many branches evolved from a single program may exist.

In PostgreSQL, as new requirements emerge, Active Developers will organize a team to implement them, in a similar way as the SRA-PostgreSQL team that implements the internationalization part. However, these new implementation will exist for a relative long time as patches, and are incorporated into the core version only after they are approved by the Core Members.

Jun evolves also as a single-version tree. As is true of many OSS development projects, there are many branches of test versions created for internal usage [18]. When the Project Leader decides that the system has been sufficiently tested, the tested version is released as a public version. In the case of Jun, every two to four internal versions are released as public.

4.3.2 Evolution of the Communities

The evolution of an OSS community is brought about by the role changes of its members. As community members change the roles they play in the community, they also change the social dynamics, and reshape the structure, of the community.

Unlike a project member in a software company whose role is determined by the project leader and remains unchanged for a long time until the member is promoted or leaves, the role that an OSS member plays in the community might be in a constant change depending on how much the member wants to get involved in the whole community. The role is not pre-assigned, and is assumed by the member as he or she interacts with the other members. An aspiring and determined member can become a Core Member through the following path.

New members are attracted to an OSS community because the system can solve one of their own problems. As Michael Tiemann put it, the depth and richness of good OSS systems often drives motivated members to want to learn more, to read the system [17]. The new members now migrate from Passive Users to Readers. As they gain more understanding of the system, they are able to fix the bugs that are either encountered by themselves or reported by others. They may also want to add a new twist to the system to make the system more powerful and more suitable for their own task. As their developed programs are made publicly available to other community members, their roles as Bug Fixers and Peripheral Developers are recognized and established in the whole community. The more contribution they make, the higher recognition they earn, and finally, they will make into the highly-selected inner circle of Core Members.

The above path describes an abstract and idealized model of role changes of aspiring members, which is common in all the four OSS projects. Not all members want to and will become Core Members. Some are always Passive Users, and some stop in the middle. Most of the users served by the four projects at SRA remain Passive Users. Members of the GNUWingnut project and the leader of the SRA-PostgreSQL project have become Active Developers due to their long-term contributions to their respective communities. Members of the Linux Support project at SRA remain to be Peripheral Developers because they have not contributed too much back to the whole community. Because all the development of Jun is conducted in SRA by the Project Leader and Core Members, the evolution of the community is limited to the outside 3 layers of Figure 1: from Passive Users to Readers to Bug Reporters.

The evolution of an OSS community is thus determined by two factors: the existence of motivated members who aspire to play roles with larger influence, and the social mechanism of the community that encourages and enables such individual role changes. This is consistent with the community-based learning theory called Legitimate Peripheral Participation (LPP) [10]. In the LPP theory, a community of professionals evolves by reproducing itself when peripheral new members (i.e. apprentices) become fully qualified members (i.e. masters). The process of becoming a master is a process of learning. In the path from a peripheral member (Passive Users in OSS communities) to a full member (respected developers), the community member acquires the skills and knowledge embodied in the community by interacting with master members (reading their codes and asking them questions about OSS systems) and practicing authentic yet small tasks (fixing bugs and developing real programs).

Free access to source code grants new members of an OSS community (1) the legitimate access to expertise of mature software development practice embodied in the system, and (2) the opportunity of working together with master developers. At

first, new members learn by engaging, peripherally, in small and easy tasks, such as fixing a bug, writing a device driver. As they become more competent and undertake more difficult tasks, they move gradually toward the center of the community and eventually establish their identities as masters in the community.

The LPP theory provides one explanation of what motivates people to get involved in OSS development—because they want to learn, and guides us how to design OSS communities that encourage and enable such learning to take place. Individual learning efforts that take place amid the interactions of the community members lead to the change of the role and the influence of that individual member, and thus the change of the whole community.

4.3.3 The Co-Evolution of Systems and Communities

For an OSS system to have a sustainable development, the system and the community must co-evolve. A large base of voluntarily contributing members is one of the most important success factors of OSS systems. As common in all the four studied projects, the evolution of an OSS community is effected by the contributions made by its aspiring and motivated members. Such contributions not only transform the role and influence of their contributors in the community and thus evolve the whole community, but also are the sources of the evolution of the system. The opposite is also true. Any modification, improvement, and extension made to an OSS system—whether it is a bug fix, a bug report, or a patch—not only evolves the system itself but also redefines the role of those contributing members and thus changes the social dynamics of the OSS community. For example, the leader of the SRA-PostgreSQL project has earned his Active Developer status by continual contributions, including the internationalization of the system.

Without new members aspiring to become a Core Member through continuous contributions to the system and the community, the development of the system will stop at the day when some of the original Core Members decide to leave the community or simply stop contributing their free time, energy, knowledge, and services any more due to some reason. Because members participate in the OSS development voluntarily, such a danger always exists. GIMP (Gnu Image Manipulation Program at http://www.gimp.org/) is such an example. When the original two creators, Peter Mattis and Spencer Kimball, left college to take jobs, they cut their tie with GIMP because they thought they had done their services to the OSS community and wanted to move beyond [8]. Because almost the whole system (95%-98%, estimated by Kimball) was developed by the two developers and at that time there was not a GIMP developer community to pick up immediately where the two left, the system stayed incomplete for more than a year. The development was resumed later when a community was finally formed.

Because the evolution of OSS communities and the evolution of OSS systems are mutually dependent, it is essential to the long-term success of OSS development that enough attention should be paid to the creation and maintenance of a dynamic and self-reproducing OSS communities. Project Leaders and Core Members of an existing OSS community should not only focus on the evolution of the OSS system itself, but also strive to create an environment and culture that fosters the sense of belongingness to the community and mechanisms that encourage new members to move toward the center of the OSS community through continual contributions.

5. THREE TYPES OF OSS PROJECTS

Based on our case study and available research literature on OSS development and evolution, we have found that at least three different types of OSS projects exist. According to its primary goal, an OSS project can be *Exploration-Oriented*, *Utility-Oriented*, or *Service-Oriented*.

5.1.1 Exploration-Oriented OSS

This type of OSS, represented by GNU software and the Jun library, aims at pushing the frontline of software development collectively through the sharing of innovations embedded in freely shared OSS systems. This is very much similar to the culture of scientific research community in which scientific results are shared through conferences and journals for peer justification, mutual inspiration, and continued development [2]. In the world of software, source code, which is the embodiment of its developer's understanding of the real world or innovative ways of changing and designing the world through software systems, is the scientific results to be shared. Due to its free access, it can spark the ideas of other developers so that something new may grow that otherwise would not have been born, and it enables others to go further by stepping on the shoulders of the previous developer through reusing the open source code [3]. For example, the Jun library represents its original developer Atsushi Aoki's unique understanding and perspective on how to model and handle 3D objects in computer displays. This library has inspired the development of several advanced research applications that directly reuse the library, as well as a new 3D library that uses the implicit model underlying Jun as the reference model [1]. Similarly, Richard Stallman started the GNU development as a way of sharing his insights in writing good programs with others.

Due to the epistemic nature of this OSS type, the quality requirements are often very high. Once the system is released publicly, it becomes a learning resource for thousands of software developers. Therefore, this type of software must be developed and maintained by expert programmers, such as Project Leaders, who often are the original developers and keep a tight control over the system in order to maintain the integrity of the system so that it reflects its original design goal. Contributions made by the community at large exist as feedback and are incorporated only if they are consistent with the ideas of the Project Leader (Figure 2a). Most community members collaborate with the Project Leader as reviewers and testers, who occasionally provide feedback.

Contrary to Eric Raymond's classification [16], the control style of this OSS type is more close to the Cathedral style than to the Bazaar style. The success of such OSS projects depends greatly on the vision and leadership of the Project Leader. However, when the vision of the Project Leader conflicts with the needs of the majority of the OSS community members, forking might happen. A new OSS project and community will be spun off the original one and embark on a similar but different project. Two typical examples are the spin-off of EGCS from GCC, and XEmacs from Emacs.

5.1.2 Utility-Oriented OSS

This type of OSS, represented by the Linux operating system (excluding the Linux kernel, which is Exploration-Oriented), aims at filling a void in functionality. Most of such OSS systems consist of many relatively independent OSS programs, such as the device drivers in the Linux operation system, and those OSS programs are developed because the original developers cannot find an existing program that fulfills their needs completely.

Rather than waiting for others to provide the needed functionality, capable software developers decide to develop their own systems and put it on the web for others to share.

As typified in the process model of the Linux Support project at SRA (Section 3.2), few OSS programs in Utility-Oriented OSS projects are completely developed from scratch. Most developers search the Internet for a partial solution and then modify it to their own needs. Their primary concern is not to use the source code as a way of scientific exploration as the Exploration-Oriented OSS developers do, but to create a program that can solve their personal needs, or scratching their personal itch [16]. Because the development is motivated by an, often emergent, practical need, timeliness is of essential importance. Moreover, because the development is driven by an individual need, developers are concerned with developing an operational system rather than delivering a refined solution as in the Exploration-Oriented type.

As the program is released for public sharing, other users who have a similar problem will pick it up, either using it as it is or modifying it further. As we have discussed in Section 3.2 in the context of the Linux Support project, the original developers are not very much concerned if they receive feedback or improvement from potential users, as long as the current program works to their satisfaction. This leads to the proliferation of programs that have similar utility. This type of OSS software development is a typical bazaar type software development. No centralized control exists. Unlike the Exploration-Oriented OSS in which forking is rather rare and the evolution of the system takes place in the form of improving the original system (Figure 2a), Utility-Oriented OSS has a lot of forks, evolving in the form of developing new programs by reusing and modifying existing programs rather than replacing the old programs with the new ones (Figure 2c). This gives to the rise of multiple, often incompatible programs. Programs that implement the similar functionality compete with each other and evolve simultaneously, but the implementation that wins the most support in the community will finally excel and eliminate other competing versions. This evolution pattern can be called as the *tournament style.*

One OSS program of the Utility-Oriented OSS itself may not have an independent community associated with it. Instead, it exists within the larger OSS community of the system of which the OSS program is a part. For example, many Linux device driver developers are a part of the larger Linux community. From the perspective of the larger OSS community, those developers of OSS programs are Peripheral Developers. Because most such developers only want to develop a program for their own particular need, they remain to be Peripheral Developers.

Because many alternatives and different versions for a piece of functionality exist, distribution packages are necessary to identify a typical set of programs chosen from a number of available alternative programs. Passive Users who just want to use the system need Readers and Peripheral Developers, who are able to match the unique needs of Passive Users to the right system, to help them choose the right configuration.

5.1.3 Service-Oriented OSS

This type of OSS, represented by the PostgreSQL system and the Apache Server [12], aims at providing stable and robust services to all the *stakeholders* of OSS systems. We use the term stakeholder to refer to both the member of the OSS community who uses the system and the end-user whose work relies on the system developed by OSS community members with the OSS system. For example, the stakeholders of PostgreSQL include PostgreSQL users and the customers of application database systems developed with PostgreSQL.

In a Service-Oriented OSS system, because the population of stakeholders is much larger than that of the OSS community, any changes made to the system have to be carefully considered so that they do not disrupt its provided services on which many end-users rely on. Therefore, Service-Oriented OSS is usually very conservative against evolutionary and rapid changes.

In accordance with its conservative nature, the control style of Service-Oriented OSS is neither Cathedral-like nor Bazaar-like. Although the Cathedral style has a tight control over the system, it is often controlled by one Project Leader, whose creative idea may not reflect the best interest of all the stakeholders. On the other hand, the Bazaar style encourages too many rapid changes to provide stable services. As we can see in the PostgreSQL community, Service-Oriented OSS is often collectively controlled by a group of Core Members, and there is no single Project Leader. Any changes are subject to debate in the group and only the changes that won the majority of the group are incorporated. We call this kind of control the Council style.

Although the control over the development of the OSS system is still centralized in the Council style, it is not controlled by any individual person. The Council is the assembly of the Core Members, who earn their rights by long-time devotion and contributions to the OSS community. Furthermore, the membership of the Council is not fixed. Most OSS communities of this type have a mechanism of accepting new council members whose contribution and competence is well recognized and who is trusted by community members.

Most members of Service-Oriented OSS communities exist as Passive Users, with some of them may become Bug Reporters and Bug Fixers as they report or submit bug fixes back to the Core Members (Council). Active Developers emerge when some big changes are needed, such as the requirements of dealing with the Japanese language in PostgreSQL. Active Developers often work with other Peripheral Developers and Bug Fixers to develop a patch for the new requirements, and the patch is finally incorporated into the major version of the system if it has been widely tested and approved by Core Members (Figure 2b).

6. DISCUSSION

We do not mean that the three types described in the above section cover all the OSS projects. Our attempt at defining three types of OSS is to create a general understanding that although all these systems are called Open-Source, differences exist in primary objective, control style, system evolution pattern, community structure and evolution pattern (See Table 1 for a summary).

6.1 Understanding Differences of OSS Projects.

Recognizing the differences of different types of OSS projects help OSS developers to identify their projects with a particular type and to take appropriate measures to guide the management and operation of the OSS project.

For example, the Project Leader of an Exploration-Oriented OSS should pay extra attention to the quality and readability of the source code by enforcing strict coding, formatting, and documenting conventions so that it will attract as many Readers as possible and fully fulfill its goal of disseminating good

Table 1: Three Types of OSS Projects

Type	Objective	Control style	System evolution pattern	Community structure	Major problems	Examples
Exploration-Oriented	Sharing innovations and knowledge	Cathedral-like central control	Single branch Feedback from the community	Project Leader Many Readers	Subject to split	GNU systems Jun Perl
Utility-Oriented	Satisfying an individual need	Bazaar-like decentralized control	Multiple versions coexist Tournament style	Many Peripheral Developers Peer support to Passive Users	Difficult to choose the right program	Linux system excluding the kernel
Service-Oriented	Providing stable services	Council-like central control	Single branch Patches merged through control	Core Members instead of a Project Leader Many Passive Users that develop systems for end-users	Less innovation	PostgreSQL Apache

programming skills and knowledge. To avoid unnecessary fragmentation of the community resources caused by forking, Project Leaders need to adapt and respond to the needs and attitudes of the community members. On the other hand, Utility-Oriented OSS projects do not need to worry too much about forking. Instead, they need to develop a social mechanism that coordinates and encourages peer support among the community members and to facilitate the easy choice of different implementations of the same functionality. Service-Oriented OSS projects should avoid being overly conservative in dealing with changes. Furthermore, they must create a social mechanism that encourages and facilitates community members to aspire to the Core Members status. Otherwise, the evolution of the system as well as the community will stop.

6.2 Evolution of OSS Projects

The type of an OSS project may evolve, as the primary objective of, and other factors affecting, the OSS project change over the time. Our working hypothesis is that Exploration-Oriented OSS and Utility-Oriented OSS are good for the initiation of an OSS project, and Service-Oriented OSS is suitable for more mature OSS projects.

Any motivated software developer can initiate an Exploration-Oriented OSS. The success of an Exploration-Oriented OSS will attract many followers, who, as users, at a certain point, will demand stability because they have invested many efforts in the OSS project and have used it to develop systems for end-users. For the benefit of the OSS project itself and the community, it is probably better for the OSS project to mutate into the Service-Oriented type; otherwise, the OSS community may split or the OSS system is simply abandoned by most community members, making the OSS project a victim of its own success.

An Exploration-Oriented OSS may also mutate into a Service-Oriented type simply because the original Project Leader has lost his or her exploration spirit, or because the system has grown too large to be controlled by any single person.

One example that has successfully completed the mutation from an Exploration-Oriented OSS to a Service-Oriented OSS is the Tcl project (http://www.scriptics.com/). Tcl was initially created in 1988 by John Ousterhout who wanted to explore a different style of system programming through the creation of a scripting language for "gluing" existing applications [15]. More than a decade later, Tcl is now used by thousands of companies and over 500,000 users, often for mission-critical applications. In the earlier years, Tcl was developed as Exploration-Oriented OSS, with Ousterhout deciding which extensions should be included based on his own interest and feedback from the large community. Since August 2000, the development of Tcl is at the helm of the Tcl Core Team (TCT) that includes 14 members in addition to John Ousterhout. The 14 members were elected by the Tcl community in recognition of their long-time devotion and support of Tcl. As other OSS projects mature, we believe that more and more such mutation will happen.

Utility-Oriented OSS projects can also mutate into Service-Oriented ones. Developers of competing implementations for a similar functionality can join forces to create a team to develop a system collaboratively that accommodate the different needs of each developer. The Apache project was started in this fashion, although the project, since it assumed the name of Apache, has

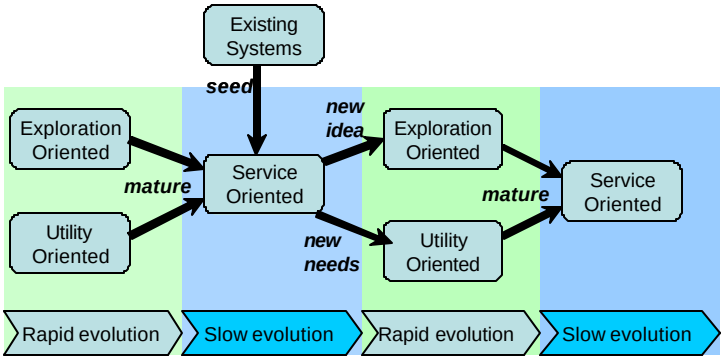

Figure 3: The Evolution Pattern of OSS Projects

been a typical Service-Oriented OSS one under the control of the Apache Group, which started with 6 Core Members and now have 25 [12]. However, the original Apache Group was formed because its members felt the need to combine their overlapping extensions and bug fixes developed individually for the NCSA httpd developed by Rob McCool, who left the system behind [4].

It is rather difficult to initiate a new Service-Oriented OSS project from scratch. Service-Oriented OSS projects are easier to take off if working systems exist already. Without a working system, few users would be interested and a community cannot form. An existing working system can serve a seed for further growth, and new requirements and ideas will be inspired as users start using the system. For example, PostgreSQL is derived from an existing research prototype.

Figure 3 hypothesizes a possible sustainable evolution pattern of OSS projects. Exploration-Oriented and Utility-Oriented OSS projects experience rapid evolution, mostly in linear fashion [7]. As the projects mature into Service-Oriented ones, the speed of evolution will slow down to a stable growth. A new round of rapid evolution will start again if the stable OSS system inspires new ideas or new requirements, giving birth to New Exploration-Oriented or Utility-Oriented OSS projects, which will again mature into new Service-Oriented ones.

This process is similar to the biological evolutionary process. According to Maturana and Varela [11], changes are determined by the structure of an organism and a perturbation. A perturbation itself does not determine how the organism evolves, but it triggers the organism to change its structure. The evolved organism with its new structure affects the outer environment and produces another perturbation. This iterative process of the interaction between the organism's structure and the environment through a perturbation is a driving force of evolution.

We have observed and reported such an evolution pattern in the Jun project [1]. Although Jun is primarily Exploration-Oriented, because it is a library on which several applications have been built, it is also concerned with providing stable services to those application developers. However, to fully document this hypothesis illustrated in Figure 3, we need to study more OSS projects for a longer term.

7. SUMMARY

As Open-Source gains popularity, many software development approaches are labeled Open-Source simply because one property of the outcome is Open-Source. However, objectives, motivations, collaboration models, system evolution patterns, and community structures and evolution patterns differ significantly from projects to projects. Rather than focusing on the common features of all Open-Source projects, this paper attempted to systematically treat the differences of OSS projects by examining the evolution patterns of systems and communities. Through analyzing a case study of four typical OSS projects, we proposed three types of OSS projects. Understanding the difference of the different types of OSS projects is the first step to build a theory that can guide the development of OSS projects.

ACKNOWLEDGEMENTS

This research is partially supported by Information-processing Promotion Agency (IPA), Japan. We thank the project members who participated in our case study. We thank Jonathan Ostwald for his profound comments and suggestions on earlier versions of this paper.

REFERENCES

[1] Aoki, A., K. Hayashi, K. Kishida, K. Nakakoji, Y. Nishinaka, B. Reeves, A. Takashima, and Y. Yamamoto. A Case Study of the Evolution of Jun: An Object-Oriented Open-Source 3D Multimedia Library, in Proceedings of 23rd International Conference on Software Engineering (ICSE'01) (Toronto, Canada, 2001), IEEE Press, 524-533.

[2] Brown, J.S., and P. Duguid. The Social Life of Information. Harvard Business School Press, Boston, MA, 2000.

[3] DiBona, C., S. Ockman, and M. Stone. eds. Open Sources: Voices from the Open Source Revolution. 1999, O'Reilly & Associates: Sebastopol, CA.

[4] Fielding, R.T. Shared Leadership in the Apache Project. Communications of the ACM, 1999. **42**(4): 42-43.

[5] FSF, GNU Coding Standards, at http://www.gnu.org/prep/standards_toc.html, accessed on 2/11, 2002

[6] FSF, GNU Philosophy, at http://www.gnu.org/philosophy/philosophy.html, accessed on 2/11, 2002

[7] Godfrey, M., and Q. Tu. Evolution in Open Source Software: A Case Study, in Proceedings of 2000 International Conference on Software Maintenance (San Jose, CA, 2000).

[8] HackVan, S., Where Did Spencer Kimball and Peter Mattis Go?, at http://devlinux.com/, accessed on 2/11, 2002

[9] Kemerer, C.F., and S. Slaughter. An Empirical Approach to Studying Software Evolution. IEEE Transactions on Software Engineering, 1999. **25**(4): 493-509.

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

[11] Maturana, H.R., and F.J. Varela. The Tree of Knowledge: The Biological Roots of Human Understanding. Shambhala Publications, Boston, MA, 1998.

[12] Mockus, A., R. Fielding, and J. Herbsleb. A Case Study of Open Source Software Development: The Apache Server, in Proceedings of 2000 International Conference on Software Engineering (ICSE2000) (Limerick, Ireland, 2000), 263-272.

[13] O'Reilly, T., OSI Business Support, at http://www.opensource.org/advocacy/case_for_business.html, accessed on 2/11, 2002

[14] O'Reilly, T. Lessons from Open-Source Software Development. Communications of the ACM, 1999. **42**(4): 33-37.

[15] Ousterhout, J. Scripting: Higher Level Programming for the 21st Century. IEEE Computer, 1998(March).

[16] Raymond, E.S., and B. Young. The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. O'Reilly, Sebastopol, CA, 2001.

[17] Tiemann, M. Future of Cygnus Solutions, in Open Sources: Voices from the Open Source Revolution, Stone, M. (ed.), O'Reilly, Sebastopol, 1999, 71-89.

[18] Torvalds, L. The Linux Edge. Communications of the ACM, 1999. **42**(4): 38-39.