Toward an Understanding of the Motivation of Open Source Software Developers

Yunwen Ye[1,2]
[1]*Department of Computer Science*
University of Colorado
Boulder, CO80309-0430, USA
yunwen@cs.colorado.edu

Kouichi Kishida[2]
[2]*SRA Key Technology Lab*
3-12 Yotsuya, Shinjuku
Tokyo 1600-0004, Japan
k2@sra.co.jp

Abstract

An Open Source Software (OSS) project is unlikely to be successful unless there is an accompanied community that provides the platform for developers and users to collaborate. Members of such communities are volunteers whose motivation to participate and contribute is of essential importance to the success of OSS projects. In this paper, we aim to create an understanding of what motivates people to participate in OSS communities. We theorize that learning is one of the motivational forces. Our theory is grounded in the learning theory of Legitimate Peripheral Participation, and is supported by analyzing the social structure of OSS communities and the co-evolution between OSS systems and communities. We also discuss practical implications of our theory for creating and maintaining sustainable OSS communities as well as for software engineering research and education.

1. Introduction

The wide success of Open Source Software (OSS) has recently attracted much attention. Software engineering researchers and commercial companies alike have been trying to learn lessons from the success of OSS and apply some of them to the development of proprietary and closed systems [1, 4, 12]. There are abundant lessons in OSS to be discovered and learned, but there is still very limited, if any, understanding of what motivates so many software developers to dedicate their time, skills, and knowledge to OSS systems with no monetary benefits.

Many definitions about OSS exist. In this paper, we define Open Source Software as those systems that give users free access to and the right to modify their source code. OSS grants not only developers but also all users, who are potential developers, the right to read and change its source code. Developers, users, and user-turned-developers form a *community of practice*. A community of practice is a group of people who are informally bounded by their common interest and practice in a specific domain. Community members regularly interact with each other for knowledge sharing and collaboration in pursuit of solutions to a common class of problems. An OSS project is unlikely to be successful unless there is an accompanied community that provides the platform for developers and users to collaborate with each other. Members of such communities are volunteers whose motivation to participate and contribute is of essential importance to the success of OSS projects.

This paper describes a conceptual framework to analyze the motivational issues in OSS. It argues that *learning is one of the major motivational forces* that attract software developers and users to participate in OSS development and to become members of OSS communities. The argument is grounded in the learning theory—*Legitimate Peripheral Participation (LPP)*, developed by Lave and Wenger [10]. The essential idea of LPP is that learning is situated in social situations, and learning takes place when members of a community of practice interact with each other in their daily practice.

The paper is organized as follows. To lay the foundation, Section 2 discusses the roles and structure of OSS communities in general, and Section 3 uses an example for further illustration. Section 4 introduces the theory of LPP. Section 5 elaborates our theory that learning is one of the motivational forces. Section 6 discusses the practical implications of our proposed theory, followed by a summary.

2. OSS Communities

The right to access and modify source code itself does not make OSS projects different from most "Closed Source Software" ones. All developers in a project in any software company would have the same access privilege. The fundamental difference is the *role transformation* of the people involved in a project. In Closed Source Software projects, developers and users are clearly defined and strictly separated. In OSS projects, there is no clear distinction between developers and users: all users are potential developers. Borrowing terms from programming languages, if we think of *developers* and *users* as types, and *persons* involved in a project as data objects, Closed Source Software projects are static-binding languages in which a *person* is bound to the type of *developer* or *user* statically, and OSS projects are dynamic-binding languages in which a *person* is bound to the type of *developer* or *user* dynamically, depending on his or her involvement with the project at a given time.

2.1. Roles in OSS Communities

The distinct feature of *role transformation* in OSS projects leads to a different social structure. People involved in a particular OSS project create a community around the project, bounded by their shared interest in using and/or developing the system. Members of an OSS community assume certain roles by themselves according to their personal interest in the project, rather than being assigned by someone else. Our previous research studying four different OSS projects has found that a member may have one of the following eight roles [13].

Project Leader. The 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.

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 those OSS projects that have evolved into their second generation, a single Project Leader no longer exists and the Core Members form a council to take the responsibility of guiding the development, such as the Apache Group and the PostgreSQL core group.

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

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

Bug Fixer. Bug Fixers fix bugs that either they discover by themselves or are reported by other members. Bug Fixers have to read and understand a small portion of the source code of the system where the bug occurs.

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. The existence of many Bug Reporters assures the high quality of OSS, because "given enough eyeballs, all bugs are shallow. [16]"

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. Given the high quality of OSS systems, some Readers read the systems to learn programming. Another group of Readers exists who read an OSS system not for the purpose of improving the system per se but for understanding its underlying model and then using the model as a reference model to implement similar systems [1].

Passive User. Passive Users just use the system in the same way as most of us use commercially available Closed Source Software. They are attracted to OSS mainly due to its high quality and the potential to be changed when needed.

Not all of the eight types of roles exist in all OSS communities, and the percentage of each type varies. Different OSS communities may use different names for the above roles. For example, some communities refer to Core Members as Maintainers. The difference between Bug Fixers and Peripheral Developers is rather small because Peripheral Developers might be mainly engaged in fixing bugs.

2.2. 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 the roles they play. Figure 1 depicts the general layered structure of OSS communities, in which the role closer to the center has a larger radius of influence. In other words, the activity of a Project Leader affects more members than that of 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 important roles 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]. Metaphorically speaking, those Passive Users play a role similar to that of the audience in a theatrical performance who offers values, recognition, and applause to the efforts of actors.

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 in the percentage of each role in the whole community. In general, 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 systems are developed by a small number of developers [12, 14].

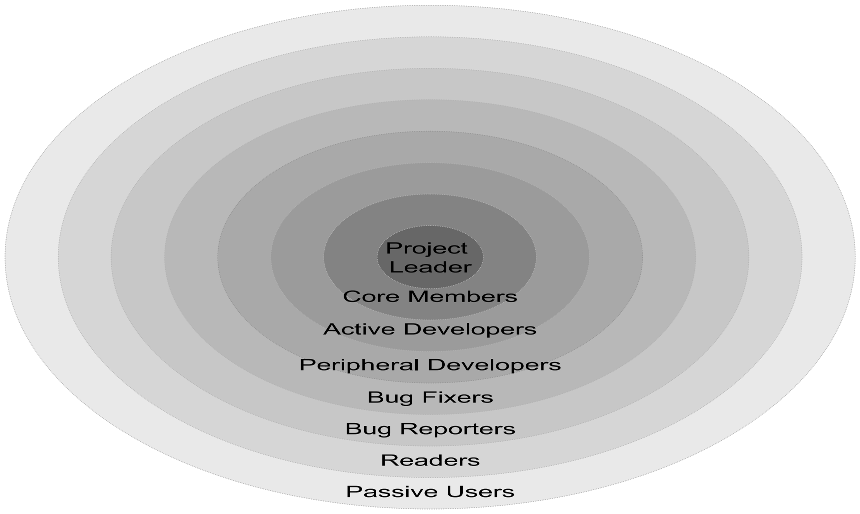

Figure 1: General structure of an OSS community

2.3. Co-evolution of OSS Systems and OSS Communities

The roles and their associated influences in OSS communities can be realized only through contributions to the community. Roles are not fixed: members can play larger roles if they aspire and make appropriate contributions. As members change the roles they play in an OSS community, they also change the social dynamics, and thus reshape the structure, of the community, resulting in the evolution of the community itself.

For an OSS project 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. 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 but also redefines the role of the contributing members and thus changes the social dynamics of the OSS community (Figure 2).

Unlike a project member in a software company whose role is determined by managers 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 constantly change, depending on how

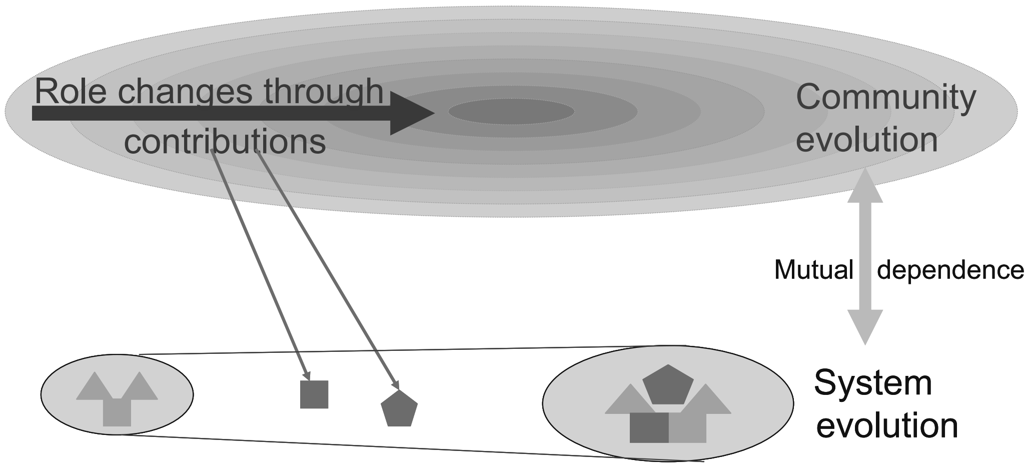

Figure 2: The co-evolution of OSS systems and OSS communities

much the member wants to get involved in the whole community. The role is not preassigned, and is assumed by the member as he or she interacts with 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. The depth and richness of good OSS systems often drives motivated members to want to learn more, to read the system [18]. The new members now migrate from being Passive Users to being 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 tasks. 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 contributions they make, the higher recognition they earn, and finally, they will enter the highly selected "inner circle" of Core Members.

The above path describes an abstract and idealized model of role changes of aspiring members. Not all members want to and will become Core Members. Some are always Passive Users, and some stop somewhere in the middle. The important point is that Open Source Software makes it possible for an aspiring and technically capable software developer to play a larger role through continual contributions. On the other hand, for OSS projects to sustain, their communities have to be able to regenerate themselves through the contributions of their members and the emergence of new Core Members and Active Developers. Otherwise, the development of projects will stop when current active contributors leave. Because all OSS developers are volunteers who are not bound by any kind of formal contracts, they may leave at any moment for various reasons.

3. An Example—The GIMP Project

In this section, we use the GIMP (Gnu Image Manipulation Program, http://www.gimp.org) project as an example to illustrate the different roles in and the structure of an OSS community, as well as the co-evolution of OSS systems and OSS communities.

3.1. The GIMP System

GIMP is a system that processes images in Linux. The system was initially created by two students, Spencer Kimball and Peter Mattis, at the University of California at Berkeley. They released version 0.51 to public on Nov. 29, 1995. This version had a plug-in mechanism for other developers to add new features easily. It was not stable at that time, but because it was one of the earliest graphics manipulation programs in Linux, it attracted many users, some of whom became developers by contributing plug-ins and helping stabilize the system. A mailing list called GIMP-Developer was soon created to discuss and share those extensions. On June 9, 1997, Kimball and Mattis released version 0.99.10, which was their final release because they graduated, started full-time employment, and no longer had time to be involved in GIMP development. The absence of Project Leaders halted further development of GIMP for about 20 months. Finally, Federico Quintero, who had developed the Color Gradient Editor for GIMP since version 0.60, assumed the role of Project Leader by coordinating the efforts of other developers and making formal release, until he left for other OSS projects. At that time, the GIMP community had well developed, and a team of Core Members, who had made major contributions to the system for a long time, formed to control and coordinate the development effort.

3.2. The GIMP Community

Like most OSS communities, the GIMP community is a virtual community. It relies on the GIMP-Developer mailing list for interested users and developers to discuss the development and use of the system, to report bugs, and to submit patches for bug fixes and new features. Analyzing the emails sent to the mailing list is one way of understanding the structure of the community. 7525 emails were sent to the mailing list between August 31, 1999 and August 02, 2002, and 913 members sent at least one email. The largest number of emails a person has sent is 817, and 502 members have sent only one email. The member who sent the most emails is currently in charge of release. Table 1 shows the compositional structure of the GIMP community based on the email traffics in the GIMP-Developer mailing list.

Table 1: The frequency of emails sent by members to the mailing list

No. of emails	No. of members	Total no. of emails
> 200	5	2237
101 - 200	8	1197
51 - 100	10	695
26 - 50	29	1061
11 - 25	47	741
5 - 10	73	471
3 - 4	107	352
2	134	267
1	502	502
Total	915	7525

Table 2: Number of code contributions and the defined roles of contributing members

No. of contributions	No. of contributors	Breakdown of the contributors according to their defined roles in the GIMP community				
		Core Members	Active Developers	Peripheral Developers	Bug Fixers	Bug Reporters
>250	3	3	0	0	0	0
101-250	4	0	4	0	0	0
51-100	11	0	10	1	0	0
21-50	15	1	12	2	0	0
3-20	47	0	21	26	0	0
1-2	82	0	0	82	0	0
Not credited in change log	25	0	0	0	10	15
Total	**197**	**4**	**47**	**111**	**10**	**15**

Table 2 displays the number of code contributions made by members to the GIMP system and the defined roles of those contributing members. We counted the number of contributions made by each person by analyzing the change log of the system. 162 persons are formally credited for the development of the system core, and about half of them (82) made only one or two contributions. The bulk of the system has been developed by 18 persons, who each made more than 50 contributions and combined to contribute more than 71% of the changes of the code. However, because counting only the number of contributions does not differentiate the importance and quality of contributed code, the number itself, albeit an important indication, does not define the roles of contributors that are recognized by the community.

The GIMP community has a quite clear definition of roles. As we have mentioned before, because the two persons who initiated GIMP have left, all GIMP developers are the second generation and four Core Members are in charge of guiding the development and are responsible for public release; 47 Active Developers are granted write access to directly contribute to the source code tree; and 111 Peripheral Developers exist whose contributions must be integrated into the released systems by Core Members or Active Developers. 476 emails are directly related to bug reports and fixes and involve 106 members, among which 10 Bug Fixers and 15 Bug Reporters are not credited in the change log. It is possible that some of the Peripheral Developers are primary Bug Fixers, but we could not find data to differentiate their roles. Core Members are the most influential figures in the GIMP community because they are the final decision makers; Active Developers can directly modify the system, and the contributions made by Peripheral Developers and Bug Fixers must be approved by Active Developers or Core Members before they are integrated.

Table 3: The co-relation between community activity and code contribution in the GIMP project

ID	No. of emails	No. of contributions	Defined role	Role transformation process
g1	817	1244	Core Member	In charge of release after 2000/12
g2	601	111	Active Developer	One of the earliest contributing members
g3	345	95	Active Developer	Granted write access in 2001/1 after contributing 10 patches
g4	245	33	Peripheral Developer	Numerous patches
g5	229	32	Peripheral Developer	Patches; responsible for a separate project Gimp-Print, a print plugin for both GIMP, Ghostscript and other systems
g6	184	5	Peripheral Developer	Numerous patches
g7	179	70	Active Developer	Granted write access in 1999/9 after contributing 7 patches
g8	155	1038	Core Member	Become active developer in 1998/2 after contributing 10 patches, become core in 2001/11
g9	141	69	Active Developer	Granted write access in 1999/9 after contributing 6 patches
g10	140	60	Active Developer	Not active since 2001/6

3.3. The Co-Evolution of GIMP System and Community

Table 3 reveals the co-relation between the active participation in the GIMP-Developer mailing list and the contributions made to the GIMP system by showing the code contributions made by the 10 most active participants in the mailing list. It is easy to see that all of them are involved in the development of the system, and 7 of them are Core Members or Active Developers.

Table 3 also illustrates the role transformations of community members, as discussed in Section 2.3. For example, `g1` and `g8` were trusted to the Core Member status due to their continuous contributions to the system and the community. Most Active Developers have earned their status of directly committing source code after they had made many contributions in the form of submitting patches as Peripheral Developers (see `g3`, `g7`, `g8` and `g9`).

4. Legitimate Peripheral Participation

Before we proceed to argue that learning is one of the motivations that attracts many users to become active contributors and drive them to contribute more to OSS systems, we need to understand how learning takes place in communities of practice by introducing the theory of *Legitimate Peripheral Participation* developed by Lave and Wenger [10] based on their studies of several communities of practice.

Communities of practice embody knowledge that is often tacit in the practice of and interactions among competent practitioners. Schön describes such knowledge as *knowing-in-action*, which practitioners demonstrate spontaneously and intuitively in their action and reaction to the constantly changing context but are unable to describe [17]. Because knowing-in-action is highly situated in the context in which it is demonstrated, learning cannot be thought of as a process of gaining, through instruction, a discrete body of abstract knowledge that learners will then transport and reapply in later contexts. Instead, learning in community should be viewed as an integral constituent of participation in the community of practice, as a process of constructing knowing through social interaction with other members of the community, of changing relationships with other members of the community, and of transforming roles and establishing identities from a journeyman to a master in the community.

LPP locates learning in the transformation of roles learners assume in the participation of community practice. Learners experience learning not as a result of being taught, but through direct engagement in the social, cultural, and technical practice of the community. Learners' entering a community entails their *legitimate participation* in real practice as collaborative partners of more competent practitioners. During collaborative participation, learners are granted legitimate access to the knowing of masters that can only be observed and understood within its context. Due to the limited capability of learners, their full participation is impossible. At first, they can only *peripherally participate* in small and easy tasks. Through legitimate participation as peripheral participants, learners create their own learning curriculum by developing a global view of the community and what there is to be learned. Learning unfolds during the interaction and collaboration with not only the masters but also other learners. As learners gain more knowledge, they become competent in undertaking more important roles, resulting in changing relationships with other members and transforming their roles in the community. Gradually, they move toward the center of the community and eventually establish their identities as competent masters in the community.

LPP refers both to the role transformation and identity development of individual members from learners to masters, and to the reproduction and evolution of the community. A community is a social unity defined by the coupling relations among its members, practice, and the outside world. The ontogenetic development of the identity of new members—from entrance as learners to becoming masters with respect to new learners who also become masters over time—changes their relations with the community as well as the relations among other members, resulting in the evolution, or the phylogenic development, of the community. The identity of the community is conserved and reproduced through the ontogenetic development of its new members who learn via legitimate peripheral participation and become masters that embody the mature practice and structural characteristics of the community.

The intertwined relationship between the ontogenetic development of the identity of an individual member and the conservation of the identify of community provides an anchor to understand the altruistic behaviors that so many OSS developers have demonstrated by contributing their time and knowledge for the benefit of the whole community. By establishing their own identities or shaping the identities of others through voluntary participation in the community practice, members help reproduce and preserve the community. This process is also in their own interests because their identity, skills, and reputation as master rely on the continuous existence of the community. Therefore, from the perspective of the community to which each member belongs, an individual's altruistic behavior is "altruistically" selfish and "selfishly" altruistic [11].

5. Learning as the Motivation

Without software developers who are motivated to start and contribute to OSS projects, OSS projects cannot succeed. Factors that affect motivation are both intrinsic (cognitive) and extrinsic (social). The precondition for motivating developers to get involved in OSS projects is that they must derive an intrinsic satisfaction in their involvement in OSS projects. Relying purely on altruism makes OSS unsustainable. Intrinsic motivation is positively reinforced and amplified when social structure and conventions of the community recognize and reward the contributions of its members.

Raymond has postulated that "scratching a personal itch" is the intrinsic motivation for OSS developers [16]. Although many developers get involved in OSS development due to the need for functionality, many OSS developers are not motivated by utility only. For example, neither Kimball nor Mattis, who started the GIMP project, had any graphic arts needs. They did not start the project because they wanted to use it [7].

We argue that learning is one of the driving forces that motivate developers to get involved in OSS projects because it provides the intrinsic satisfaction for OSS developers, and the role transformation in OSS communities that go along with learning offers the extrinsic motivation.

Software systems are cognitive artifacts whose creation is a process of knowledge construction that requires both creativity and a wide variety of knowledge about problem domains, logic, computer, and others. In this sense, software systems, like books, are a form of knowledge media. Many OSS systems come into existence as results of the learning efforts of their original developers who try to understand how to model, or to change, the world with computational systems, as we will explain in Section 5.1. When the source code become accessible to users, the knowledge and creativity therein also become accessible, providing the initial learning resource that attracts users to form a community of practice around the system. By participating in the community, developers and users learn from the system, from each other, and share their learning with each other, as we will discuss in detail in Section 5.2.

5.1. Learning Experience of OSS Initiators

Initiators of new OSS projects may be motivated by *explorative learning* or *learning by doing*. These two forms of learning are not mutually exclusively; they may exist simultaneously to inspire the initiation of an OSS project.

Explorative learning. This form of learning is similar to most scientific research in which learners (e.g., scientists, practitioners) attempt to find new ways of doing things or of overcoming an existing problem. OSS systems are viewed by GNU developers as "scientific knowledge to be shared among mankind" [6]. Larry Wall started Perl because he ran into a problem he couldn't solve with existing tools, and he wanted to explore a way of doing things better [20]. Similarly, John Ousterhout initiated his Tcl/Tk project because he wanted to create a reusable tool command language for his many research systems [15]. Atsushi Aoki started Jun—a Smalltalk and Java library for manipulating 3D graphics and multimedia data, because he wanted to explore the possibility of handling both geometry and topology of 3D graphics [1].

Learning by doing. In this form of learning, the learners want to deepen their understanding of a certain domain by actually engaging in practical tasks that allow them to apply their existing knowledge and to perfect their current skills. By definition, hackers, who are behind almost every OSS systems, are people who enjoy "exploring the details of programmable systems and how to stretch their capabilities" through programming rather than theorizing [1]. Linus Torvalds started Linux partially because he wanted to learn more about the architecture of Intel 386, and the perfect way of doing so was to develop an operating system for it [3]. Peter Mattis described his "original impetus for GTK was simply (his) wanting to understand how to write a UI toolkit" [7].

5.2. Learning Experience of Later Participants

When the results of the above, more often than not, individual learning efforts are distributed in the form of open source, they provide resources and opportunities for other developers to learn. Most developers who start an OSS project are master programmers, and their systems are the products of fine craftsmanship and examples of excellent programming practice. When those systems are freely distributed, they grant developers in the world the *legitimate* access to the skill and knowledge embedded in such systems. Similar to the way that we learn to write by reading literature, reading existing source code of expert programmers is a powerful path to the mastery of programming art. We believe that many developers are attracted to OSS projects because they want to learn something. As Michael Tiemann, founder of Cygnus, put it: "It was this depth and richness that drove me to want to learn more, to read the GNU Emacs Manual and the GNU Emacs source code. [18]" We also believe that OSS participants learn a lot from their OSS experience. As Patrick Volkerding, creator of the Slackware distribution of Linux, commented: "My experience with Linux has taught me a lot of valuable skills. It looks like the project has saved me from a life of COBOL. What more could I ask for than that? [8]"

The learning experience of later participants of OSS projects does not stop at passive absorption by reading source code; it also happens when new participants engage in bug reports, maintenance, and further development of OSS projects. In most cases, new

[1] Jargon Dictionary, available at http://info.astrian.net/jargon/terms/.

participants do not become Core Members suddenly. As we have analyzed in Section 2.3 conceptually and illustrated in detail with the GIMP example in Section 3, they have to earn their status and recognition in the community gradually by making small contributions at first. In other words, they start with *peripheral participation* by, for example, reporting and fixing bugs. By doing so, they learn by doing and their skills improve. As their skills are gradually recognized in the community based on their contributions, they are trusted to bigger and more challenging tasks, and move toward the “inner circle” of the community, becoming competent full participants and exerting larger influence (Figure 2).

Active participation of new members creates opportunities for them to interact with other more knowledgeably skilled developers, and gives them *legitimate* access to the expertise therein. The existence of OSS communities enables new participants to ask questions about a variety of aspects of the OSS systems and to acquire help in using, understanding, modifying, and extending the systems. One study on the communication patterns of the FreeBSD Newconfig project has found that 18.8% of the emails in the mailing list of the project are questions, and 49.9% of the emails are responses that include answers to questions, agreement, disagreement, and additional information [21]. Questions and answers are also very common in the GIMP-Developer mailing list. Such communities create social networks of knowledge in which all participants share their knowledge and learn from each other.

Another major function of the mailing list is to provide a platform for developers to discuss the functionality, design, and implementation of the system. In other words, the mailing list displays the ongoing process of creating the product. The process is yet another learning resource for newcomers, who are given the opportunity not only to see how the system is developed as it is being developed, but to understand the culture of the community and make it theirs by observing how skilled developers talk, work, and collaborate with each other.

5.3. Social Aspects of Extrinsic Motivations

The social fabric inherent in OSS communities reinforces the intrinsic motivation for participating in OSS projects as a form of learning. Only in a society where technical supremacy is highly appreciated can developers acquire good reputations among their peers by displaying their skills through free distribution, and often wider acceptance, of their systems. The good reputation attracts attention, trust, and cooperation from others and lays the foundation for advancing the original developers’ agenda and the establishment and development of OSS communities.

Members close to the center of the community (Figure 1) enjoy better visibility and reputations than do peripheral members. The road to the core has to be paved by contributing more to the project and interacting more with their members (Figure 2). As new members contribute to the system and the community, they are rewarded with higher recognition and trust in the community, and higher influence as well. In the GIMP community, most developers who have contributed a lot are given the right to directly contribute to the system. Some even become Core Members.

Rewarding contributing members with higher recognition and more important roles is also important for the sustainability of the community and the system development, because it is the way that the community reproduces itself (Section 4). In the GIMP community, 29 Active Developers have not been active for at least a year, but the community is still prospering because many new developers have become competent participants along the path of LPP. From the log of source code commitments, we have found that 25 developers started contributing code in the recent two years.

5.4. An Oriental Perspective

The viewpoint of learning as a motivation that intrinsically drives people to get involved in OSS development and that extrinsically rewards them with higher social status and larger influence in OSS communities is in parallel with a tradition of Eastern culture. Intellectual property is a very new concept in Eastern culture; instead, scholars have long pursued intellectual prevalence by commanding high recognition and respect from the people, especially the ruling class and intelligentsia, through the free distribution of their writings. Writings are treated as the heritage and public assets of the whole society and they are free to all. More importantly, all writings are open to interpretation. In fact, most scholars build their own theory and knowledge by commenting and annotating the writings of earlier scholars while they are reading. Although comments and annotations are often the products of the scholars’ own efforts of understanding, assessing, and learning the writings produced by others, they become free learning resources and inspire further modifications and interpretations. The hallmark of an established scholar is the authority of interpreting the writings of a well-respected ancient scholar (e.g. Confucius), and only those who can integrate the ideas of their ancestors and contemporaries alike and convince others with their freely distributed writings can acquire such status.

6. Discussions

Realizing that learning is one of the major driving forces for OSS development has practical implications in managing OSS projects and raises several questions in software engineering education and research.

6.1. Creating Opportunities for LPP

Learning through LPP is not a result of direct and intentional teaching; instead, it is enabled by legitimate participation in practice and legitimate access to learning resources—products and process—available in the community. The openness of the produced system, the development process, and the communications among members in OSS communities enables learning by watching and invites learning by doing, and thereby is directly related to the learning experience of the people involved [9]. Although all OSS communities are open to certain forms of participation and access, the different control structure inherent in each OSS community due to considerations of system quality [13] creates different degrees of openness that allows the legitimate participation and access of community members.

Table 4 shows the possible combinations of openness in two dimensions: product (row) and process (column). In the product dimension, *open release* means that only formally released versions are accessible to all community members; and *open development* means that all interim developing versions are accessible. In the process dimension, *closed process* means that the discussion of system development is conducted mostly within the "inner circle", often through a strictly controlled mailing list which is not accessible to other members; *transparent process* means that although only the "inner circle" is involved in the development process, but their discussion is readable by other members; *open process* means that the development decision is conducted in public space, allowing the participation and access of all interested parties. To encourage learning-motivated participation requires the highest degree of openness in both dimensions because it offers more learning resources. However, it may also reduce the Project Leader's control over the system. This conflict needs to be balanced by those who want to make their systems open source.

The possibility for newcomers to participate peripherally is another key point in LPP. To attract more users to become developers, the system architecture must be designed in a modularized way to create many relatively independent tasks with progressive difficulty so that newcomers can start to participate peripherally and move on gradually to take charge of more difficult tasks. The way a system is partitioned has consequences for both the efficiency of parallel development—a prerequisite to OSS—and for the efficiency of knowledge acquisition. This adds an extra dimension of importance to the modularity of software systems because it ensures the possibility of legitimate peripheral participation of new members. The plug-in architecture of GIMP is quite effective in engaging new developers. Linux could not be such successful without its well designed modularity [19].

Table 4: Openness of OSS communities

	Open release	**Open development**
Closed process	GNU; Jun	APACHE
Transparent process	Tcl/Tk	PostgreSQL
Open process		GIMP

Another approach to afford peripheral participation is to intentionally under-design the system by leaving some non-critical parts unimplemented to facilitate easy participation. The TODO list most OSS systems have creates guidance for participation. Rather than just listing TODO items, grouping them according to their estimated difficulty might provide a better roadmap for newcomers to start participation at periphery.

6.2. Advice for OSS Practitioners

Developers at the center of OSS communities should not only focus on the development of the system itself, but also pay enough attention to the creation and maintenance of a dynamic and self-reproducing OSS community. Core Members and Active Developers should also strive to create an environment and culture that fosters the sense of belonging to the community and mechanisms that encourage and enable newcomers to move toward the center of the community through continual contributions. It is very important for the community to be responsive to the questions and contributions of newcomers to sustain their interest and encourage their further participation. Old members should remember that they are also the learning resources for newcomers. One possible mechanism is to have skilled members take turns in the mailing list to answer questions of newcomers, to help new contributors perfect their code contributions.

People who want to start an OSS project need to consider how many learning opportunities it offers, and how easy it is for others to participate legitimately and peripherally. For example, a system developed with COBOL is probably less attractive to OSS developers than a system developed with Java because the demands for COBOL developers are much lower in today's world. A system with large size and cumbersome architecture, such as the early version of Mozilla, is also difficult to attract OSS participants [2].

6.3. Impacts on Education and Research

The existence of many OSS projects provides a possibility for educators to change the way of educating and training new software professionals in schools [5]. By integrating OSS projects with university classes, students are given the chance to learn programming by reading the existing systems developed by world-class professionals. At the same time, they can work together with skilled developers and gain practical experience in developing systems of industry scale and strength. From such collaborations, students also acquire the communication skills of presenting ideas effectively and of taking feedback from other developers, which are essential in practical development settings that invariably require teamwork. Moreover, introducing students into OSS projects will foster the next generation of OSS developers and sustain the further development of the OSS movement.

Given the importance of learning in OSS communities, the importance of the skill of reading programs and tools that support program reading and understanding should be stressed more [1]. Current software engineer education and research focuses mostly on system writing. However, as we all know from our language learning experience, to become good writers, we have to learn to read first, and read a great amount of good literature.

7. Summary

In this paper, we have tried to create an understanding of what motivates people to participate in OSS development. We applied the LPP theory to understand how to form and sustain OSS communities that are essential to the success of OSS projects. We argued that learning is one of the major driving forces that motivate people to get involved in OSS communities. We discussed how our theory can inform software engineering researchers and OSS practitioners.

OSS is a very complicated phenomenon that is related to technology, human behaviors, economics, culture, and society. We do not claim that our theory clarifies all those complicated, intertwined relationships, but we do believe that it creates a better theoretical understanding of OSS from a new analytical angel, and provides practical guide to the management and development of OSS projects.

Acknowledgment.
The authors thank Kumiyo Nakakoji and Yasuhiro Yamamoto for their valuable feedback. The research was supported by the National Science Foundation, Software Engineering and Language Program, Grant CCR-0204277.

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] Baker, M., *The Mozilla Project and Mozilla.org*, at http://www.mozilla.org/editorials/mozilla-overview.html, accessed on 2/11, 2002

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

[4] Dinkelacker, J., P.K. Garg, R. Miller, and D. Nelson. "Progressive Open Source," in *Proceedings of 24th International Conference on Software Engineering (ICSE'02)* (Orlando, FL., 2002), ACM Press, 177-186.

[5] Dvorak, G. "Collective Education," *Ubiquity*, 2001. **2**(16): http://www.acm.org/ubiquity/views/g_dvorak_1.html

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

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

[8] Hughes, P. "Interview with Patrick Volkerding," *Linux Journal*, 1994. **1994**(2es): 3.

[9] Hutchins, E. *Cognition in the Wild*. The MIT Press, Cambridge, MA, 1994.

[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 Publicaions, 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] Nakakoji, K., Y. Yamamoto, Y. Nishinaka, K. Kishida, and Y. Ye. "Evolution Patterns of Open-Source Software Systems and Communities," in *Proceedings of International Workshop on Principles of Software Evolution (IWPSE 2002)* (Orlando, FL, 2002), 76-85.

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

[15] Ousterhout, J. "Scripting: Higher Level Programming for the 21st Century," *IEEE Computer*, 1998. **31**(3): 23-30.

[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] Schön, D.A. *The Reflective Practitioner: How Professionals Think in Action*. Basic Books, New York, 1983.

[18] Tiemann, M. "Future of Cygnus Solutions," in *Open Sources: Voices from the Open Source Revolution*, DiBona, C., S. Ockman, and M. Stone. (eds.), O'Reilly, Sebastopol, 1999, 71-89.

[19] Torvalds, L. "The Linux Edge," *Commun. ACM*, 1999. **42**(4): 38-39.

[20] Wall, L. "The Origin of the Camel Lot in the Breakdown of the Bilingual Unix," *Commun. ACM*, 1999. **42**(4): 40-41.

[21] Yamaguchi, Y., M. Yokozawa, T. Shinohara, and T. Ishida. "Collaboration with Lean Media: How Open-Source Software Succeeds," in *Proceedings of 2000 International Conference on Computer Supported Cooperative Work (CSCW'00)* (Philadelphia, PA, 2000), ACM Press, 329-338.