Understanding OSS Openness through Relationship between Patch Acceptance and Evolution Pattern

Passakorn Phannachitta[†] Pijak Jirapiwong[‡] Akinori Ihara[†]
Masao Ohira[†] Ken-ichi Matsumoto[†]

[†]Nara Institute of Science and Technology
8916-5, Takayama, Ikoma
Nara, Japan
{phannachitta-p, akinori-i, masao, matumoto}
@is.naist.jp

[‡] Kasetsart University
50 Ngam Wong Wan Rd, Chatuchak
Bangkok, Thailand
b5005135@ku.ac.th

ABSTRACT

Openness is referred how much does the OSS core committer team share and compile with their non-committers. Because the openness can be varied from time to time, a study for explaining the vary of openness would be a good approach to support the OSS. It will not only encourage the non-committers to exert more contribution when the openness is high, but it will also make the them still have an optimistic outlook on the project when the openness is low. Unfortunately, there is only a few studies aiming to understand this principle. This work, we seek out for the key factors that identify the transitory changed of the openness. Unlike most previous studies, we focus on the clear relevant evidences that are more concrete. Our temporal-based analysis on patch acceptance in two major OSS projects: Apache HTTP Server and Eclipse Platform conclude that special event occurrences have a decisive influence over the openness either in transitory or lasting. Moreover, our investigation on the relationship between the temporal changes of openness and a plausible OSS evolution pattern broadens the mutual accord in both openness and the evolution of OSS.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging; D.2.9 [**Software Engineering**]: Management

General Terms

Management, Measurement, Experimentation

Keywords

Open Source Software, OSS Evolution Pattern, Temporal Factor, Openness, Patch Submission, Patch Acceptance

1. INTRODUCTION

An active collaboration between every role in Open Source Software (OSS) project is one of the most important key for the sustainable development [6]. Many OSS supportive activities will be ardently achieved whenever each role is well-balanced in the number of participant and everyone exerts the best effort. For an example on patch submitting activity, which committers and non-committers are the two main different roles. Non-committers whose authority makes them unable to apply patches for the project directly will be the patch submitter, and the committers will be the judge who accept the patch. An OSS project needs a high number of active non-committers to increase and widen the perspective of requirements and patches the covers the larger area of defects. The OSS committer team should also have enough members that is capable for verifying all the incoming requirements and patches thoroughly. Ideally the committer should always be open and shared to the non-committer so as to encourage them to increase their contribution.

Nowadays, a study that aims for comprehension between committers and non-committers is quite overlooked. Most studies give more attention to the individual supportive either for committers or the non-committers. Studying the OSS committer's openness can be one that support both committers and non-committers at the same time. Although it has been concluded that temporal changes of the openness are existence [14], more deepening study for the varieties of the temporal factors is noteworthy. Analyzing without concerning the temporal factors, the broader insightful study of the OSS committer's openness such as the open season prediction or answering why the openness level is always changed will be inconceivable.

This research aims to explain the transitory changed of the committer's openness. First, we have analyzed the patch acceptance in temporal on the two well-known OSS projects, Apache HTTP Server and Eclipse Platform. Temporal factor is important since it's unfair to conclude how open of an OSS committer team from the summarized statistic. From our preliminary results, we have found that the special event occurrences were the key factor influenced to the transitory openness level. Further, we have noticed that there are some remarkable treads exposed to the vary of the openness. Elucidating those trends we have found the relationship between the temporal changed of openness with a plausible OSS evolution pattern proposed by Nakakoji et.al [7]. The relationship we found is not only a sounded explanation for understanding the changes of the openness, but also a proof of the validity of our chosen evolution pattern.

The remainder of the paper is organized as follows: Section II explains the backgrounds and briefs the existing related works. Section III introduces our research questions. Section IV explains our methodology. Section V describes the case study setup and discusses on our finding. Section VI contains an observation from the study. And finally, section VII provides the conclusion and outlines some future works.

2. BACKGROUND KNOWLEDGE

2.1 Related Works

To date, it has been quite lack of studies aimed to improve the comprehension between committers and non-committer. Most researches provide useful guidances for each individual role. Bettenburg et al. suggested non-committers what will make a good bug report [1]. Weißgerber et al. also suggested them the proper size of the submitting patch [13]. On the committer side, Wang et al. and Runeson et al. help the committer ascertain if the reported bug is duplicated [9,12], and Rigby et al analyze the personality of the top committer [8].

Nonetheless, a few studies proposed to make the comprehension between both roles, which will support both the same time. Shibuya et al. have studied the process of participating in OSS [10]. Their finding encourages the non-committer to gain more reputations; however, their work is still lacking in enthusiasm for making committer be more open to the non-committer.

2.2 Patch Submission and Patch Acceptance

In OSS, patches contain different information between two versions of the same file. Exchanging patches instead is very convenience since the receiver can definitely know the changed information without any further attempt. In practical, when a non-committer needs some changes with a target file (i.e. A file that locates in the project repository) , that non-committer will check out the target file and edit it. After that, the edited target file will be converted into a patch and sent to the committers. Patches are usually sent through a common channel provided by the OSS community. The currently most popular channel is bug-tracking system such as Bugzilla. Along the way to the committers, the bug-tracking will let the submitted patch be discussed. It helps the committer to evaluate how good and valuable of the submitted patch. At last, if that patch reach a consensus and the committers concur, the committers will apply the submitted patch with the target file and commit it. Whether the whole patch or just its portion is committed, it means the committers accept the patch.

There is an interesting phenomenon that the committers can gradually accept a patch. Because a large patch would contain more components which probably have high dependencies. It seems impossible that all the components would be accepted and committed at once. Unless we include the gradual patch-acceptance case, we will be unable to report the reflected result from the patch acceptance analysis. Unfortunately, the recently existing researches ignored this phenomenon [2, 13] that may lead the authors to an inaccurate conclusion.

2.3 Openness and Open Season

Openness is a term referred how much do the committers share and comply with their non-committers. To date, it is still inconclusive that a high openness committer is always a good committer [8]. However it's a rational thought that as long as the non-committers know the committers are open, they will give their earnest effort to the project, because the non-committers would feel they are important know that they are not being isolated. Moreover, avoiding the decreasing of openness will induce the committers to give more attentions to the non-committers.

Openness can be analyzed in many aspects, such as counting the committers' reply messages or approximately accumulating their spending time with the bug report. However, an absolutely clear evidence such as the number of patch acceptance would rather conclude the openness. In this work we can know the actual committed number of a patch (from the partially committed case). We define an openness in our works by counting the time that committer's committed any part of patches in a defined period. It would give us more accurate insight than counting the number of accepted patches directly. There is an interesting question about analyzing the openness about concerning the temporal factor Definitely, an OSS project has been continually and unceasing developed. It's hard to believe that the committers team is composed with the same people throughout the time. The team can be reorganized so that the openness is possible to be changed. Since the openness is changeable, patches can also be more or less accepted in some periods. We will denote a period contains a relative high number of patch acceptance as in opening season.

2.4 Evolution Pattern of OSS

The temporal-based analysis of the OSS committers' openness will inherently reveal the characteristics of OSS project distinctively between each development period. A further analysis from these informative characteristic will broaden the accord about an evolution of OSS. Currently, there are several existing studies of OSS evolution patterns [3, 4, 11], we selected one plausible pattern proposed by Nakakoji et al [7]. They categorized OSS project into three types, and also outline the characteristics of each type in many dimensions Three types of OSS project are Exploration Oriented, Utility Oriented, and Service oriented. We summarize the dimensions related to our work in table 1.

2.4.1 *Exploration Oriented*

An OSS project usually becomes an Exploration Oriented when the project itself has been received some new ideas. Normally, new ideas are emerged by the senior developers or the project leader whom know the project very well so that the new ideas are usually discussed among the core members. Being an Exploration Oriented project, core developers or committers seem to more closed comparing between other periods. It's more likely to be happened when the project was just founded or after it has been mature for a while.

2.4.2 *Utility Oriented*

An OSS project usually becomes this type when a project needs of some new features or enhancements. It's different from Exploration Oriented, because requiring for a new feature or enhancement can be a requirement from anyone. Reforming into a Utility Oriented is possible anytime, either the project is under developing and has a plenty of bugs or the project has been being mature for a long time. Being a Utility Oriented project, core developers or committers seem to be most opened among other periods, since they need more suggestions to be discussed.

2.4.3 *Service Oriented*

An OSS project is usually reformed into Service Oriented after a project released a stable version. The goal of a Service Oriented project is to provide service as stable as possible. Being this type, core developers or committers are slightly less open than a Utility Oriented project and the

Table 1: Three Types of OSS Project

	Exploration-Oriented	Utility-Oriented	Service-Oriented
Objective	Sharing Innovations and Knowledge	Satisfying an individual need	Providing stable service
Control style	Cathedral-like central control	Bazaar-like decentralized control	Council-like central control
Community structure	Project leader and core members	Core members and many peripheral developers	Core members and Many passive users
Major problems	Finding a novel innovation	Difficult to choose the right program	Less innovation

openness is likely to be continually decreased, because the project has been more stable that would normally have less number of defects.

From the three classified types of the OSS project proposed by Nakakoji et.al [7], they believed that the evolution pattern of any OSS will be alternating from one type to others as illustrated in figure 1. They also gave some examples of an idea for the transition between each types (i.e. When a Service-Oriented project has some new needs, it will reform into a Utility-Oriented project.) However the conclusion to support if the evolution pattern is respected to this pattern as shown in the figure 1 are still waiting for a proof.

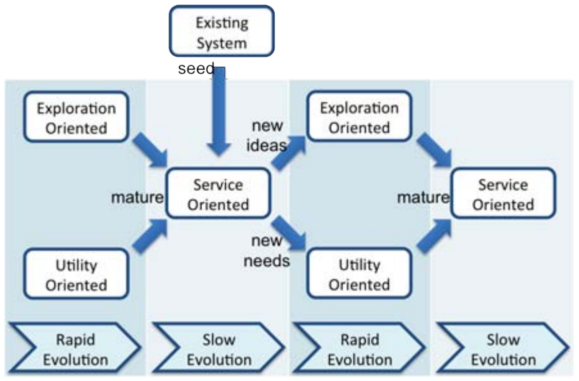


Figure 1: The Evolution Pattern of OSS Projects [7]

3. RESEARCH QUESTION

To develop an understanding of the transitory changed of OSS committers' openness through analyzing the patch acceptance and OSS evolution pattern, we need to answer two research questions.

3.1 What is the key factor that makes the OSS committers changed their openness?

Tackling this question, we would understand clearer about the patch acceptance. Achieving the key factor that induce an open season will allow us to briefly foretell the non-committers when should they exert more contribution. Furthermore, if the open season characteristic is likely to be in common, it could be a feature for predicting the open season, which is more promising. On the other hand, we also need a sound reason to explain why do the committers decrease their openness in some periods. Without a sounded reason, the non-committer may consider that the committer teams are having some trouble that they are unable to thoroughly verify the submitted requirements and patches.

3.2 How can an OSS evolution pattern explain the trend of OSS committers' openness?

Unlike the other routine works, the openness of the OSS committers does not always stay at the same level [14]. At least it still respected with some trends (i.e. keeps increasing or decreasing for a period of time). OSS evolution pattern may have an supportive explanation since it explains the change of OSS projects. If it is existed, we would reach a better insight of the OSS openness and the patch submission activity.

4. METHODOLOGY

For the case study, we develop a method to gather the patch acceptance traces divided into period of time. The method has three phases that are Patch extraction, Diff file creation, and Patch acceptance identification.

4.1 Patches extraction

We have two types of data source; Mailing list and Bugzilla. Mailing list is where committer and non-committer discuss about patch and bug, and Bugzilla is a well-known bug tracking system. Their structures are totally different, so we need two specific patch extraction methods.

For the mailing list, we choose to improvise and extend the Weißgerber et al.'s proposed [13]. It is the closest method that fulfilled our requirement in patch extraction from email. On the other hand, we have to develop a specific web site crawler to extract patches from Bugzilla from the scratch since such an explicated proposed method is nonexistent.

In both methods, we denote our patch format with a tuple $(I_p, P_p, t_p, L_p, [c_p])$ where I_p is the patch's index, P_p is the patch's absolute path that we can identify the corresponded target file in the repository, t_p is the patch's submitting time, which we has already converted into a common timezone (UTC) for avoiding an incorrect time-stamp identification. L_p is the total number of the changed lines of code, and $[c_p]$ is a list contained all the individual changed lines of code. Note that, we collapse all white spaces in $[c_p]$, because in practical the committer may apply patches manually. It then may produce some white spaces shifted that would lead us to an inaccuracy analysis. After we has extracted all the information in $(I_p, P_p, t_p, L_p, [c_p])$ from the raw-patch data , we stored them in a database.

4.2 Diff files creation

We denote a Diff file as a file that contains the changes information between each committed version at the repository side. These changed information make us able to track whether the submitted patch are accepted. We create diff files between each adjoining revision of all the target files in the repository in order to know when do the patches are committed. Moreover, creating a diff file between each adjoining revision make us able to include the gradual patch acceptance case into our study.

In this phase, we also have two types of repository data source: CVS and SVN. For both types of repository, at first we extract the revision number and the timestamp of each target file. A pair of revision number and timestamp then guide us to obtain all the change information of target files. Next, we create diff files from each pairs of adjoining revision and then extract the required information similar to the patch extraction. We also denote the diff files as a tuple (I_r, P_r, t_r, $[c_r]$). I_r is committed source code's index, P_r is the its absolute path, t_r is its timestamp. We also parse the time zone into a common timezone as the patches. $[c_r]$ is the list of changed line in a revision which white spaces are also collapsed . At last, we compose (I_r, P_r, t_r, $[c_r]$) into records and store them in a database same as the extracted patches.

4.3 Partial Acceptance Identification and Commitment Counting

Since our analysis includes the gradually accepted case, we conclude a $patch_i$ as an accepted patch if there is an existed line of the submitted code that has been committed with the corresponded target file to the repository within a time limit Δt.

$$\begin{aligned} &(I_p = I_r \vee P_p \sim_{match} \ P_r) \\ \wedge\ & t_p\ +\Delta t \leq t_r \\ \wedge\ & (\exists l\ | l \in [c_p]) \subseteq [c_r] \end{aligned}$$

We count up the number of patch commitments of the accepted patches to study the openness. The counter of patch commitment increases each time either when a patch is fully committed or partially committed.

5. CASE STUDIES

We study on two well-known OSS projects: Apache HTTP Server and Eclipse Platform. Apache HTTP Server has been introduced as a web server since 1996 and is still popular. Eclipse Platform is a part of Eclipse project, which is a well-known interface development environment (IDE) project. We analyze the patch submission and acceptance on Apache HTTP Server between its mailing list system and its SVN repository. For another Eclipse platform project, we analyze between its bug-tracking system named Bugzilla and its CVS repository. Table 2 shows the quantitative information on both case-study data sources.

Table 2: The characteristic of the case-study datasets

(a) Repositories

	Apache HTTP Server	Eclipse Platform
Repository	SVN	CVS
Observing period	1998/06 - 2003/06	2002/06 - 2008/06
#File	6283	46,004
#Changed line	1,994,030	9,532,211

(b) Patches Data Source

	Apache HTTP Server	Eclipse Platform
BTS	Mailing list	Bugzilla
Observing period	1998/06 - 2002/06	2002/06 - 2007/06
#Patch	5,212	75,808
#Target file	1,926	71,153
#Changed line	140,391	9,919,338

We assign the time scope Δt as 365 days on both data sets because we concern the gradual patch accepted case. Larger patches are normally composed with more components so that they will probably take longer time to be fully accepted. Note that the Patches Data Sources contain one year less data than the repositories to make the experimental results reflected with our max Δt as 365 days. Figure 2 is illustrated our experimental results performed on both datasets. In both graphs the y axis denotes the total number of patch commitment, and the x axis denotes the timeline in month interval. We will explain about the label above the graph later during the subsidiary of research questions.

5.1 What is the key factor that makes the OSS committers changed their openness?

Observing through the both timelines 2(a) and 2(b), we are massively captivated by the frequent occurrences of pulses. It's rather unusual that needs a satisfactory explanation. There are several good metrics analogous to the temporal factors [5]. Special event occurrence and trend are very interesting factors. However, the pulses' appearance seems to have high impact and be in transitory that make us prefers a hypothesis about the occurrence of special events over the trend.

We have noticed two different characteristics of the pulses in both figures 2(a) and 2(b). First is the number of commitment has continually increased or decreased after a pulse is occurred. Another one is a pulse is surrounded by lower number of commitment that are approximate equivalence. Therefore, there should be more than one type of event occurred.

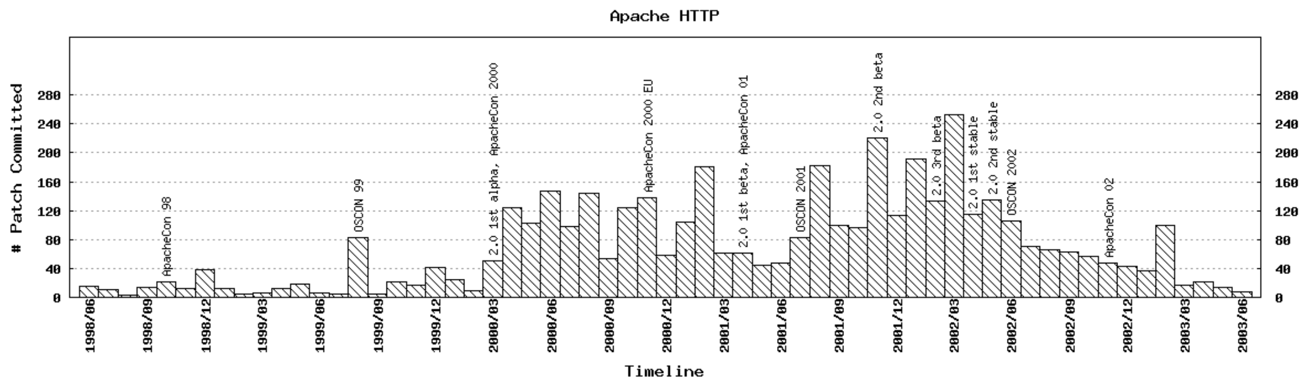
5.1.1 The number of patch commitment has continually increased or decreased after a pulse is occurred

We start a deduction from the most ordinary case. We thought the decreasing (i.e. 2003/02 in Apache HTTP Server and 2006/02 in Eclipse platform) is tend to more unusual than the increasing. Because the increasing number of patch acceptance may as usual as the project growth. One plausible explanation is during a decreasing patch commitment period, the project was stable enough and had the less number of severe bugs.

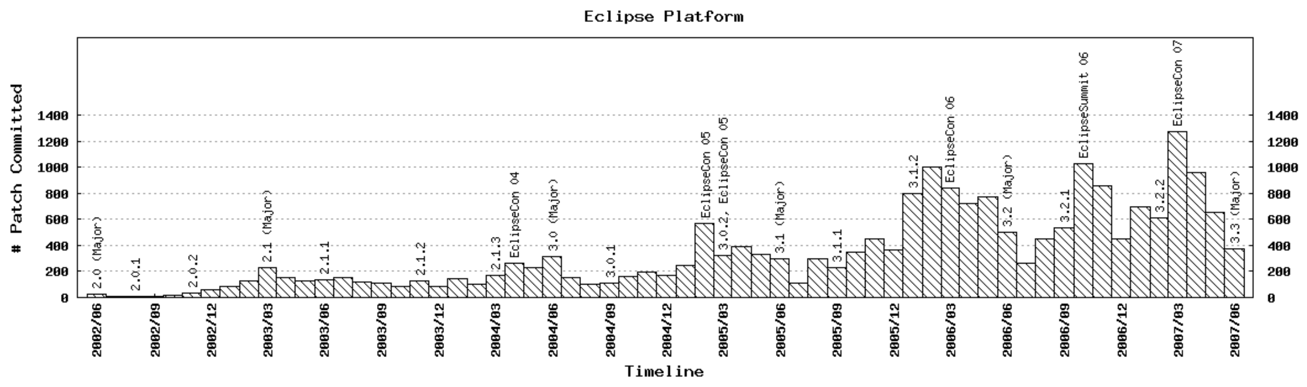
We investigated the released dates of the project in prior versions, and the released dates we have found are likely matched with the pulses related to this case. We label the corresponded released version over the graph in figure 2. It supports our hypothesis about the stable released versions. Moreover, it also tells us about the increasing period of patch commitment after a pulse is occurred (i.e. 2000/03 in Apache HTTP Server and 2005/02 in Eclipse platform). The pulses belonged to this case are matched with the release of some minor versions (i.e. alpha and beta version). Because the release of minor version still probably has many topics needed to be discussed. Note that, here the released of minor versions may not produce a pulse, but the number of patch commitment is always increased for a short period after that. (i.e. 2002/02 in Apache HTTP Server and 2002/11 in Eclipse platform)

5.1.2 A pulse is surrounded by a lower commitment periods that are approximate equivalent level

We have outlines a hypothesis about the events supporting this case. The supportive event for this type of pulse should be held in a short period and probably has a transitory effect, since it does not induce any change of the patch



(a) Analyzing Patch Acceptance from Apache HTTP Server project between June 1998 and June 2003



(b) Analyzing Patch Acceptance from Eclipse Platform project between June 2002 and June 2007

Figure 2: The Patch Acceptance Analysis Results from Apache HTTP Server and Eclipse Platform

acceptance.

We start our investigation on the first significant pulse of Apache HTTP Server in August 1999. Focusing on that time point, we turn up to a record of the first O'Reilly Apache Conference held in that month. It is perfectly matched with our hypothesis that caused by the OSS community must have placed some advertisements for the forthcoming conference. (i.e. calling for paper) It then captivated more developer to participate with the OSS community. We also track for more corresponding conference shown on the figure 2. Eclipse platform is more explicit to conclude this hypothesis with the conference occurrence.

5.2 How can an OSS evolution pattern explain the trend of OSS committers' openness?

Focus on the control style column of each type of OSS project in table 1, the three different styles tell us obviously that we can distinguish each type of the OSS projects by the amount of openness. Since patch acceptance is directly related with the openness, a relationship between OSS evolution pattern and the patch acceptance would exactly be existed. In figure 2, if we omit the special-event pulses, there will be only the trends of the patch acceptance left: increasing, decreasing, and stay-the-same.

The most simple case is the increase of patch acceptance. Since the Bazaar-like decentralized control [6] is the only one type of OSS project that could generate it, we could assume the project must have been a Utility-Oriented during the increasing of patch acceptance. We then also believe that the project was transiently Utility-Oriented during a special event pulse, because of the drastically increment of the accepted patch. Next, the decreasing trend of the patch commitment is usually happened after a major released. It is caused by the project is more stable, then fewer defects needed to be fixed. Consequently, the objective of the Service-Oriented as shown in table 1 makes it perfectly matched with this case.

However, the stay-the-same case is quite complicated. It can be occurred in several situations. The most simple case for the stay-the-same is when the project has just been founded or been well-known. This case is simply concluded as an Exploration-Oriented type. Then, what would be happened if the stay-the-same is occurred after the increasing (Utility-Oriented) period? After many needs and requirements were satisfied and many features also have been appended; the project should be more mature. We will conclude the stay-the-same after an increasing of patch acceptance period as a transition period between Utility-Oriented and Service-Oriented. At last, what if the stay-the-same was occurred after the decreasing (Service-Oriented) period. After a project has been mature for while, the need of expansion should become an issue. It may be a transition to become an Exploration-Oriented or a Utility-Oriented. If new ideas and innovations are more important, it will be a transition to be an Exploration-Oriented. On the other hand, if the expansion is inspired by new requirement, it will probably be a transition to be a Utility-Oriented.

6. OBSERVATION

Comparing between two projects, Eclipse Platform are more predictable. Figure 2(b) shows that Eclipse platform has been continual growing respected to the almost same pattern from year to year (start in March). Most of the local relative peaks are always in the month that held a conference, and the number of patch acceptance always decreases after a major release. We believed it is caused by Eclipse Platform has a clear routine on its conference (around March) and the released date of major version (around June). Since the target checkpoint is clearly defined, all the participant will know how much should they exert their effort averse to the time to make the project reached the goal. Moreover, the clearly defined checkpoint would make the project self-comparable and let the participants know how much does the project develop from year to year. It is very interesting that when Eclipse platform has introduced a conference (Eclipse Summit) in 2006, the growing pattern is still the same but has been shorten (From March to October and October to March). This is a good proof of our caprice that the occurrence of conference has an influence to the openness, especially in a project that defines the important events very well such as Eclipse platform.

7. CONCLUSION AND FUTURE WORK

In this research, we found out what makes the OSS committers changed their openness and how could an OSS evolution pattern explain it. We analyzed two well-known OSS projects: Apache HTTP Server and Eclipse Platform. The results revealed two interesting identities of the OSS committers' openness. First, we found that at least two types of special event occurrences would affect the openness. When an official conference or workshop are forthcoming, the openness will remarkably increase. The conference's announcement and advertisement would arouse the newcomers as well as the currently inactive developers and users. The project then become more active so that wider varieties of patches were produced and submitted. The second type of special event is the released versions.After a minor version was released, there usually still has some open issues waiting to be resolved so that the committers would be still or opened. Alternately, a major version should be released after most of opening issues has already been resolved; hence, the number of patch acceptance would typically be decreased.

Our second finding is the relationship between the transitory changed level of openness and the OSS evolution pattern proposed by Nakakoji et.al [7]. The relationship we found can elucidate that the alternation of committer's openness level is an effect of the OSS project's evolution. Moreover, the result from our in temporal-base patch acceptance analysis can be a proof for the inconclusive hypothesis the evolution pattern, which they believed an OSS project would be evolved alternately from a type to others continually.

Beyond this research, we will explore more aspect and study more features to develop a knowledge of OSS committers' openness. We are ardently believed that openness and open season are entirely predictable.

Acknowledgment

The first and second authors are grateful to the internship program cooperated and supported between Kasetsart University, Thailand, and Nara Institute of Science and Technology, Japan. It bestows a grant as well as an opportunity for undergraduate student to achieve a wealth experience in abroad graduated school research.

This research is being conducted as a part of the Next Generation IT Program and Grant-in-aid for Young Scientists (B), 22700033, 2010 by the Ministry of Education, Culture, Sports, Science and Technology, Japan.

8. REFERENCES

[1] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering (SIGSOFT '08)*, 2008.

[2] C. Bird, A. Gourley, and P. Devanbu, "Detecting patch submission and acceptance in oss projects," in *Proceedings of the Fourth International Workshop on Mining Software Repositories (MSR '07)*, May 2007.

[3] A. Capiluppi, J. M. González-Barahona, I. Herraiz, and G. Robles, "Adapting the "staged model for software evolution" to free/libre/open source software," in *Ninth international workshop on Principles of software evolution: in conjunction with the 6th ESEC/FSE joint meeting (IWPSE '07)*, 2007.

[4] M. W. Godfrey and Q. Tu, "Evolution in open source software: A case study," in *Proceedings of the International Conference on Software Maintenance (ICSM'00)*, 2000.

[5] B. Manaskasemsak, A. Rungsawang, and H. Yamana, "Time-weighted web authoritative ranking," *Inf. Retr.*, vol. 14, April 2011.

[6] K. Nakakoji, K. Yamada, and E. Giaccardi, "Understanding the nature of collaboration in open-source software development," in *Proceedings of the 12th Asia-Pacific Software Engineering Conference*, 2005.

[7] K. Nakakoji, Y. Yamamoto, Y. Nishinaka, K. Kishida, and Y. Ye, "Evolution patterns of open-source software systems and communities," in *Proceedings of the International Workshop on Principles of Software Evolution (IWPSE '02)*, 2002.

[8] P. C. Rigby and A. E. Hassan, "What can oss mailing lists tell us? a preliminary psychometric text analysis of the apache developer mailing list," in *Proceedings of the Fourth International Workshop on Mining Software Repositories (MSR '07)*, 2007.

[9] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of duplicate defect reports using natural language processing," in *Proceedings of the 29th international conference on Software Engineering (ICSE '07)*, 2007.

[10] B. Shibuya and T. Tamai, "Understanding the process of participating in open source communities," in *Proceedings of the 2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development (FLOSS '09)*, 2009.

[11] N. Smith and J. F. Ramil, "Agent-based simulation of open source evolution," in *Software Process Improvement and Practice*, 2006.

[12] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," in *Proceedings of the 30th international conference on Software engineering (ICSE '08)*, 2008.

[13] P. Weißgerber, D. Neu, and S. Diehl, "Small patches get in!" in *Proceedings of the 2008 international working conference on Mining software repositories (MSR '08)*, 2008.

[14] M. Yamamoto, M. Ohira, Y. Kamei, S. Matsumoto, and K. Matsumoto, "Temporal changes of the openness of an oss community: A case study of the apache http server community," in *Proceedings of The Fifth International Conference on Collaboration Technologies (CollabTech '09)*, 2009.