A Time-Lag Analysis toward Improving the Efficiency of Communications among OSS Developers

Masao Ohira, Kiwako Koyama, Akinori Ihara,
Shinsuke Matsumoto, Yasutaka Kamei, and Ken-ichi Matsumoto

Graduate School of Information Science,
Nara Institute of Science and Technology,
8916-5, Takayama, Ikoma, Nara, Japan
{masao,kiwako-k,akinori-i,shinsuke-m,yasuta-k,matumoto}@is.naist.jp
http://se.naist.jp/

Abstract. Open source software (OSS) is developed by globally distributed developers with a variety of lifestyles. In such the development environment, the time-lag of communications among developers is more likely to happen due to the time difference among locations and the difference of working hours for OSS development. A means for effective communications among OSS developers has been increasingly demanded in recent years, since even an OSS product and its users requires a prompt response to issues such as defects and security vulnerabilities. In this paper, we propose an analysis method for observing the time-lag of communications among developers in an OSS project and then facilitating the communications effectively. We have conducted a case study in which our analysis method was applied to mailing-list data of the Python project. As the results, we have confirmed that our method could identify the existence of the time-lag in communications among Python developers and have achieved findings on the optimum timing for the communications.

Key words: time-lag analysis, distributed software development, open source software, OSS community

1 Introduction

Open source software (OSS) such as Linux and Apache is generally developed by globally distributed developers. Unlike commercial software development in a company, OSS development does not necessarily request developers to engage in development at a designated time and location. OSS developers may voluntarily decide whether they continue to dedicate themselves to OSS development or not.

In this OSS development environment, a time-lag occurs in communications among developers more than a little, because of differences of time zones among geographically-distributed developers with a variety of lifestyles. For instance,

according to the geographical distribution of registered users at SourceForge[1] which was reported by Robles and Gonzalez-Barahona [1], the top three regions by the number of registered developers at SourceForge are North America, West Europe, and China. Since the time-lag among those regions is at least more than five hours, it would not be easy to discuss among developers in real-time. Furthermore, even if developers reside in the same time zone, it is not still guaranteed that developers can communicate each other in real time, because each developer has no constraint on working hours.

While the importance of decision-making and consensus building through discussions among developers is increasing especially in a large-scale OSS project with a number of developers, communications among developers with various time zones and lifestyles might trigger an occurrence of a time-lag and then impede rapid OSS development. In particular, in case prompt actions are required (e.g., fixing critical bugs regarding security vulnerability), the delay of decision-making and consensus building due to the communication time-lag among developers would results in decreasing software reliability and loosing users' trust.

The goal of our research is to construct a support mechanism for effective communications among geographically-distributed OSS developers. As a first step toward achieving the goal, in this paper we present an analysis method for helping OSS developers comprehend a whole picture of a communication time-lag occurred in a OSS project. The analysis method targets mailing list archives as a data source, and consists of three kinds of analyses as follows;

1. analysis of a geographical distribution and activity time of OSS developers
2. analysis of a distribution of time required for information exchanges among OSS developers in different locations, and
3. analysis of appropriate timing for sending messages.

From a case study with Python project [2] data, this paper explores the usefulness of the analysis method.

2 Analysis Method

This section describes data extraction, conversion and classification which are necessary in advance of performing our analysis.

2.1 Preparation

Data extraction and conversion. The target data source for our analysis is archives of mailing lists which are used by OSS developers to exchange information. The reason we select mailing list archives as the target data for our analysis

[1] SourceForge is one of the largest OSS development community, which provides registered projects with a variety of software development support tools such as source code management tool, bug tracking system, and mailing lists. As of February 2009, more than 230,000 OSS projects and more than two million users have been registered to SourceForge.

(a) a thread in a mailing list

post A : _____
- **reply B** : Re:_____
  - **reply C** : Re:Re:_____
- **reply D** : Re:_____

(b) a list of information of posted/replied messages

ID	posted date and time	location (timezone)	Coordinated Universal Time (UTC)
message A	2009/04/29 10:33:53	UTC＋9	2009/04/29 01:33:53
message B	2009/04/28 22:40:04	UTC－4	2009/04/29 02:40:04
message C	2009/04/29 09:12:30	UTC＋3	2009/04/29 06:12:30
message D	2009/04/29 02:26:59	UTC－10	2009/04/29 12:26:59

(c) a relationship among posted and replied messages

info. of posted messages			info. of replied messages			time lag (hours)
ID	posted date and time	location	ID	replied date	location	
message A	2009/04/29 10:33:53	UTC＋9	message B	2009/04/28 22:40:04	UTC－4	1.10
message B	2009/04/28 22:40:04	UTC－4	message C	2009/04/29 09:12:30	UTC＋3	3.54
message A	2009/04/29 10:33:53	UTC＋9	message D	2009/04/29 02:26:59	UTC－10	10.88

Fig. 1. Data extraction and conversion

is because mailing lists are widely used in OSS projects. We consider that data of mailing list archives allows us to reveal a whole picture of the existence of time-lag in many OSS projects.

In order to apply the analysis method to the target data, firstly we need to extract information of **posted date and time**, and **posted locations** from mailing list archives (i.e. from e-mail headers). In what follows, "posted date and time" means local date and time of a message's sender, and "posted locations" is presented as time-lag between Coordinated Universal Time (UTC) and local time. For instance, "**UTC+9**" means the location of Japan because the standard time of Japan is nine hours prior to UTC.

Figure 1 shows the procedure of data extraction and conversion. When a developer posts a message to a mailing list, the message is delivered to subscribed developers of the mailing list. Replying to the post, the other developers can discuss the message Using such the post-reply relationship (i.e., thread structure) in a mailing list, we extract information on posted/replied date and time, and locations (time zones) from mailing list archives [2].

For instance, from a thread structure illustrated in Fig.1(a), we extract information of posted and replied messages as the table in Fig.1(b). Then we convert the information into post-reply relationships as the table in Fig.1(c) and calculate time-lag from a difference between posted and replied date and time. Note that we suppose that message B replied to message A can be a posted message for message C.

Classification of data. Several factors such as differences of time zones (i.e., countries and/or regions) and differences of developers' working hours may have

[2] We do not collect data from posted messages with no replies.

an influence on time-lag between posted time and replied time. For instance, communications among developers living in different time zones might be prolonged because of differences of lifestyles (e.g., dinner time or sleeping time). And developers in the same time zone might be still difficult to communicate each other in real time, because each developer has no constraint on working hours.

In order to distinguish between the time-lag due to time zone differences and the time-lag due to lifestyle differences, the collected data described above is classified into data **within** and **over** the time-lag of 24 hours. Many of replied messages within 24 hours after a post would be affected by differences of time zones, while replied messages over 24 hours after a post would be generated by differences of developers' lifestyles and/or difficulty of the content of a posted message, rather than geographical differences among developers. For these reasons, our analysis method targets the data of posted and replied messages within the 24 hours time-lag.

2.2 Procedure

Geographical distribution and activity time of OSS developers. In order to understand the existence of the communication time-lag in an OSS project, the analysis method firstly identifies a geographical distribution of developers of the project, counting the number of replied messages by each location (UTC−11 ∼UTC+12). The analysis method also identifies a distribution of the number of replied messages by local time in each location in oder to understand working hours of developers by each location, since developers' working hour can differ even in the same location. By this means, we can identify active or inactive locations and working hours of OSS developers.

Distribution of time required for information exchanges among OSS developers in different locations. In order to understand the communication time-lag due to the geographical (time zone) differences, the analysis method calculates distributions of time required for information exchanges among OSS developers in **different** locations and the **same** locations respectively. This helps us more clearly distinguish between the time-lag by the geographical differences and the time-lag by the differences of developers' lifestyles.

Appropriate timing for sending messages. In order to identify the appropriate timing for communications which resolves communication time-lags as much as possible, the analysis method calculates the number of replied messages by each hour, using **posted (local) time** and **replied (local) time**. A numerical number in Fig.2(a) shows size of time-lag (hours) between time zones A and B. Fig.2(b) shows the number of pairs of posted messages from time zone A and replied messages from time zones B. For instance, suppose that one developer in A post a message between 9 and 12, and other developer in B replies a message

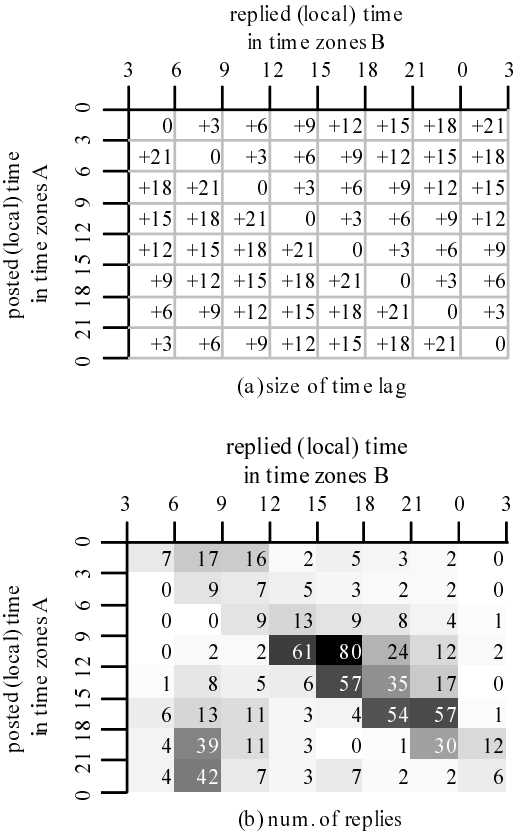

Fig. 2. Distribution of posted and replied time

between 15 and 18. In this case, the time-lag is +3 hours and the number of post/reply pairs is 80.

Time zones A and B are fixed after selecting target locations for analysis. Time zones B in Fig.2 is arranged as replied messages within an hour correspond to posted messages on the diagonal. In Fig.2, size of time-lag and the number of posted/replied messages are counted by three hours, but the length may be changed depends on analysis needs. Furthermore, the all cells in Fig.2(b) are gray-scaled according to the number of posted/replied pairs of messages, to grasp a big picture of time slots with a large or small number of replied messages.

Using Fig.2(a) and (b), it is possible to identify time slots with large or small time-lag. For instance, we can see that messages posted between 21 and 0 in time zones A (the bottom row in Fig.2) tend to be replied after 6 hours. That is, to post messages from 21 to 0 would not be the appropriate timing for less time-lag communications.

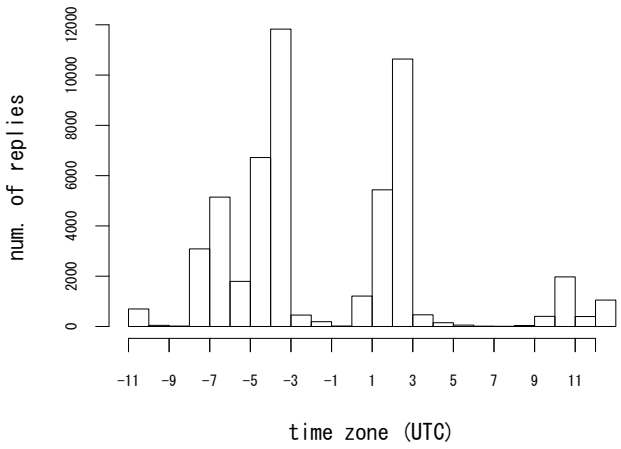

Fig. 3. Distribution of the number of replied messages by time zones

3 Case Study

This section describes a case study with a mailing list for developers in the Python project. Through the case study, we would like to confirm whether the analysis method can help us understand the existence of time-lags in communications among OSS developers.

3.1 Python

Python is an object oriented script language developed by OSS. It is very popular in Europe and the United States as well as Perl. Because it supports various platforms and provides rich documentations and libraries, it is used in a broad range of domains (e.g., Web programming, GUI-based appricaitons, CAD, 3D modeling, formula manipulation, and so forth).

3.2 Target data

We selected the mailing list archive called"Python-Dev" which is for discussing development of Python such as new features, release and maintenance. We use the Python-Dev mailing list archive from April 1999 to April 2009, which have 89,301 messages. Excluding posted messages with no replies and messages with no information on posted/replied time and locations, posted and replied messages were 56,707. 51,830 of 56,707 messages were sent within 24 hours.

Table 1. Target locations for the case study of Python

region	time zone	locations
North and South American continent	UTC-8〜UTC-4	United States, Canada, West of Brazil, Chile, Bolivia, Mexico, etc.
European and African continent	UTC+0〜UTC+3	Europe, Africa, Moscow, Iran, Saudi Arabia, etc.

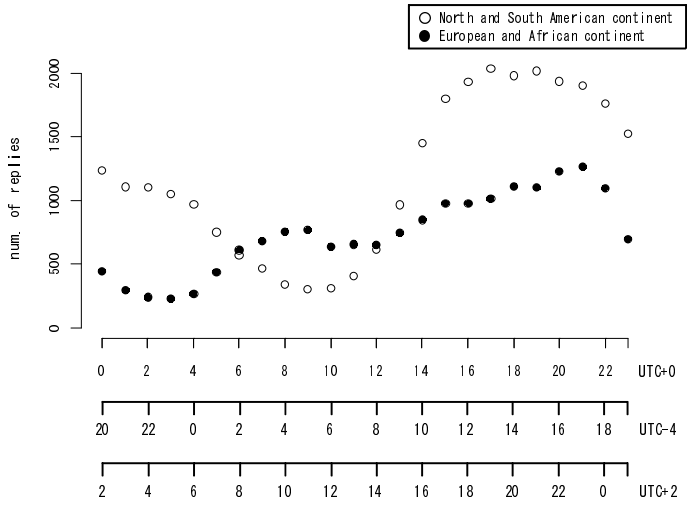

Fig. 4. Distribution of the number of replied messages by time slots (white circles: North and South American continent, black circles: European and African continent)

3.3 Analysis results

Analysis of a geographical distribution and activity time. Fig.3 shows a distribution of the number of replied messages by time zones. The X-axis and Y-axis respectively mean time zones and the number of replied messages.

Fig.3 indicates that in the Python project, a large number of messages are replied by developers from UTC-4 (East of the United States) and UTC+2 (central Europe). This result is not surprising at all. Because Python is mainly used and developed by European and American developers, it would be natural that developers living in the locations actively communicated.

Many of countries in the locations of UTC-4 and UTC+2 is utilizing daylight-saving time. And countries around the countries in UTC-4 and UTC+2 also have many messages. So, we selected two regions around UTC-4 (the North and South American continent: UTC-8〜UTC-4) and UTC+2 (the European and African continent: UTC+0〜UTC+3) as the analysis target in this paper. Table 1 shows major countries included in these regions.

Table 2. Statistics of time-lags by region (A: North and South American continent, E: European and African continent)

posted region → replied region replied region	the number of replies	maximum (hours)	median (hours)	minimum (hours)
A → A	18,901	11.55	1.24	0.00
A → E	6,942	16.34	2.07	0.00
E → E	9,426	14.69	1.59	0.00
E → A	7,215	13.91	1.80	0.00

Fig.4 shows transitions of replied messages by hour in the two regions which are determined from Fig.3. The X-axes shows time in the three time zones (UTC+0, UTC−4, UTC+2) and the Y-axis is the number of replied messages.

Fig.4 indicates that the maximum and minimum number of replied messages from the North and South American continent are attained respectively at 13 and 5 in the local time (UTC−4). Python developers in the North and South American continent seem to mainly communicate during daytime hours. In contrast, Python developers in the European and African continent actively communicate during nighttime hours, because the number of replied messages from the European and African continent is peaked at 23 in the local time (UTC+2). In this way, analyzing activity time of OSS developers by using the number of replied messages helps us understand the existence of the difference of working hours by region.

Although Fig.4 provides an overview on the difference of working hours of OSS developers by region, however, it does not tell us anything about time-lags. In fact, developers in the both regions actively communicate each other from 12 to 23 in UTC+0. Communication time-lags might not exist in the regions. In contrast, developers in either one region or the other region does not actively communicate from 12 to 23 in UTC+0. Communication time-lags between developers living different locations might exist in this time period.

Analysis of a distribution of time required for information exchanges among OSS developers in different locations. Table 2 shows time spent to reply messages to the same and different time zones, the number of replied messages, and time-lags (maximum/median/minimum). A pair of a post from location X and a reply from location Y is represented as "X → Y".

The median hours of time-lag among the same time zone was 1.24 hours for A → A and 1.59 hours for E → E. The median hours of time-lag between the different time zones was 2.07 hours for A → E and 1.80 hours for E → A. Developers in the same time zone can expect to have a reply within 90 minutes, and developers between different time zones also can expect to have a reply within about 2 hours. Since the actual difference of time-lag between the target regions is nearly 6 hours, we can consider that communication time-lags in the Python project is relatively small.

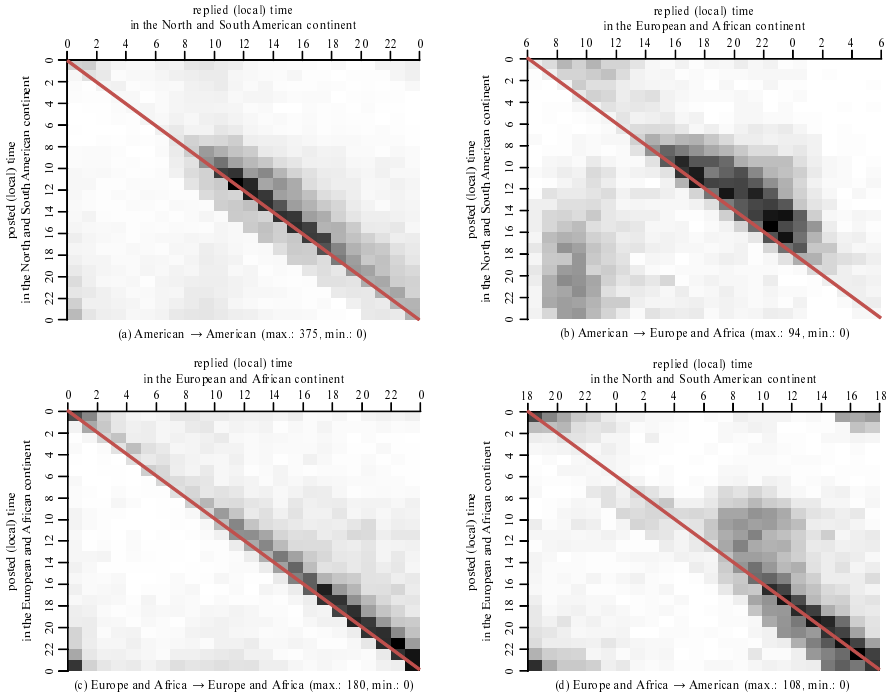

Fig. 5. Distributions of posted/replied local time between two regions

Analysis of appropriate timing for sending messages. Fig.5 (a), (b), (c) and (d) are distributions of the number of replied messages between two regions. For the simplicity, only gray-scaled figures without the number of replied messages are shown in Fig.5. We can see that the zero time-lag (i.e., dark gray cells near the diagonal line) is expected from 10 to 17 in Fig.5(a), from 9 to 17 in posted (local) time and from 15 to 23 in replied (local) time in Fig.5(b), from 16 to 23 in Fig.5(c), and from 16 to 23 in posted (local) time and from 10 to 17 in replied (local) time in Fig.5(d). For these time periods, developers would timely communicate each other.

In contrast, reply time seems to be delayed from 18 to 23 in posted (local) time in Fig.5(b) and from 7 to 13 in replied (local) time in Fig.5(d), because there are darker cells a short distance away from the diagonal line. These two posted (local) time periods correspond to the time period from midnight to early morning (0 to 6) in replied locations, which means that developers in replied locations was sleeping at the posted time.

From the result of Fig.5, in oder to receive a quick reply, it would be desirable to post a message from 10 to 17 in the North and South American continent, and from 16 to 23 in the European and African continent. On the contrary, it is not appropriate timing to post a message from 18 to 23 in the North and South American continent, and from 7 to 13 in the European and African continent,

since time-lag is likely to occur. In this way, our analysis method helps OSS developers know the appropriate timing so that they can resolve a time-lag of information exchange in an OSS project as much as possible.

4 Discussions

Opposite to what we expected before our case study, we have confirmed in Table 2 that the influence of time-lag due to the time zone difference was relatively small in the Python project. One reason of this phenomena might be that active time of Python developers is partly overlapping in the two regions. Although there are about 6 hours time-zone difference between the two regions, the active time in the North and South American continent was different from that in the European and African continent as shown in Fig.4. Therefore, active hours of Python developers in the two regions might overlap by coincidence from 10 to 17 in the North and South American continent (from 16 to 23 in the European and African continent). Another reason may be that the number of Python developers subscribed to the "Python-Dev" mailing list is sufficiently-large to quickly respond to a posted message at any time.

Our analysis method is not only useful in knowing the appropriate timing for communications among geographically-distributed OSS developers, but also useful in changing communication media used in an project. For instance, when a project replaces mailing lists with IRCs (Internet Relay Chat) as communication media, developers would be required to more precisely understand the appropriate timing for communications to resolve time-lag. In that case, our method would help developers know the better timing for real-time communications.

OSS developers are not necessary to be geographically-distributed, but they may be at the same region or location. Though our analysis method mainly aims to understand the communication time-lag arising from time-zone differences, it can be used for the time-lag due to lifestyle differences of OSS developers in the same region or location. OSS developers have no constraint on their working hours and they can freely engage in OSS development. At the same region, some developers can work in the morning and other developers can develop OSS at midnight. Depending on the differences of lifestyles of developers, time-lags could happen even if they live close to each other. In this situation, our method can provide an insight on the differences of active time in the same region and help developers understand the appropriate timing for sending messages.

The analysis method also can be used for distributed development in a company. Working hours in a company are fixed to some extent, but it is not necessarily that a developer in one site can communicate with other developers in another site at a particular time. In the prior study [3], time zone differences are visualized to understand and exploit overlapping hours in a distributed environment. Our method can not only visualize the time zone differences, but also allows developers to understand the easiness of communication at a particular time period, using the number of replied messages (i.e., density of working activity at a particular time period).

In this paper, we introduce the time-lag analysis method toward improving the communication efficiency of geographically-distributed OSS developers. The analysis method targets mailing list archive data as communication logs to reveal the existence of communication time-lags. Although IRC communications are often used in OSS projects and they can be our analysis target, communications using IRC do not work when developers one wishes to talk are off-line. So, IRC communication logs are not likely to well-capture communication time-lags.

In this paper, we have conducted a case study of the Python project, using the "Python-Dev" mailing list archive. Python-Dev consists of about 10 years mailing list archive data. So, it might be too large to show communication time-lags among Python developers at the fine-grained level. Actually, we have observed that communication time-lags in the Python project were relatively small. We suspect that this results from the size population of developers (subscribers) of Python-Dev. In Python-Dev[4], a posted messages must be read by a number of developers in the world and so it might be easy to have replies. In order to emphasize the existence of time-lags and its issues, in the near future, we need to analyze more specific situations such as the level of communications among module owners, reviewers and patch contributors.

5 Related Work

The issues on communication time-lag or delay in OSS development have been intensively studied in relation to bug modification processes with bug tracking systems in open source projects [5–15]. For instance, Wang et al. proposed several metrics to measure the evolution of open source software [14]. The metrics include the number of bugs in software, the number of modified bugs and so on. As a result of a case study using the Ubuntu project which is one of Linux-based operating system distributions, the study found that about 20% of all the reported bugs were actually resolved and over ten thousand bugs were not assigned to developers. These findings indicate that it takes a long time to resolve all bugs reported into bug tracking systems and that it also takes a long time to start modifying bugs. The study, however, did not reveal the amount of time or communication time-lags to resolve bugs.

Mockus et al. [12] and Herraiz et al. [7] have reported studies on the mean time to resolve bugs in open source software development. Mockus et al. [12] have conducted two case studies of the Apache and Mozilla projects to reveal success factors of open source software development. In the case studies, they analyzed the mean time to resolve bugs because rapid modifications of software bugs are generally demanded by users. As a result of the analysis, they have found that the mean time to resolve bugs were short if bugs existed in modules regarding to kernel and protocol, and existed in modules with widely-used functions. They also found that 50% of bugs with the priority P1 and P3 were resolved within 30 days, 50% of bugs with P2 were resolved within 80 days, and 50% of bugs with P4 and P5 were resolved within 1000 days. While [12, 7] mainly focused on precise understandings of bug modification processes in open source software de-

velopment, we are interested in the influence of communication time-lags among developers on the bug modification process.

The issues on differences of time-zone and/or geographical distance in distributed development rather have been discussed in terms of the context of corporate (proprietary) software development [16–20]. For instance, Harbsleb et al. [18] have compared single-site development with milti-sites development and then revealed that development in the distributed environment introduced the delay of development speed. In contrast, Bird et al. [21] analyized the development of Windows Vista by comparing distributed teams with collocated teams from the aspect of the post-release failures of components. They have found a slight difference in failures, but the difference have been less significant. Nguyen et al. [22] also reported the similar phenomena in the Eclipse Jazz project. Although the lessons learned from these studies on distributed software development provides us a lot of useful insights, they are partly applicable to geographically-distributed OSS development due to the differences of lifestyles of developers even in the same region or location. In this paper, we tried to tackle this unique feature of time-lags in OSS development.

6 Conclusion and Future Work

In this paper, we proposed an analysis method for observing the time-lag of communications among developers in an OSS project and then facilitating effective communications. As the results of our case study applying the analysis method to the Python developers' mailing list archive, we could confirm that our analysis method helps geographically-distributed OSS developers understand that

- active time of developers are different from regions,
- communication time-lags in the Python project is relatively small, and
- there exists the appropriate timing for resolving communication time-lags as much as possible.

In this paper, our analysis method targets communication time-lags in the two regions with the time zone difference. In the future, we need to analyze regions and/or locations without time zone differences in order to better understand the influence of lifestyle differences of developers on communication time-lags. As described before, we still need to analyze more specific situations of time-lags at the fine-grained level.

7 Acknowledgment

This research is being conducted as a part of the Next Generation IT Program and Grant-in-aid for Young Scientists (B)–20700028, 21–8995, 20–9220 by the Ministry of Education, Culture, Sports, Science and Technology, Japan.

References

1. Robles, G., Gonzalez-Barahona, J.M.: Geographic location of developers at sourceforge. In: Proceedings of the International Workshop on Mining Software Repositories. (2006) 144–150
2. Python Programming Language – Official Website: http://www.python.org/
3. Laredo, J.A., Ranjan, R.: Continuous improvement through iterative development in a multi-geography. In: 2008 IEEE International Conference on Global Software Engineering. Volume 0., Los Alamitos, CA, USA, IEEE Computer Society (2008) 232–236
4. Python-Dev – Python core developers ML: http://mail.python.org/mailman/listinfo/python-dev
5. Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R., Zimmermann, T.: What makes a good bug report? In: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering(FSE'08). (2008) 308–318
6. Godfrey, M.W., Tu, Q.: Evolution in open source software: A case study. In: Proceedings of the International Conference on Software Maintenance(ICSM'00). (2000) 131–142
7. Herraiz, I., German, D.M., Gonzalez-Barahona, J.M., Robles, G.: Towards a simplification of the bug report form in eclipse. In: Proceedings of the 2008 international working conference on Mining software repositories (MSR'08). (2008) 145–148
8. Ihara, A., Ohira, M., Matsumoto, K.i.: An analysis method for improving a bug modification process in open source software development. In: IWPSE-Evol '09: Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops, New York, NY, USA, ACM (2009) 135–144
9. Kim, S., Pan, K., Whitehead, Jr., E.E.J.: Memories of bug fixes. In: Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering(FSE'06). (2006) 35–45
10. Kim, S., Zimmermann, T., Pan, K., Whitehead, E.J.J.: Automatic identification of bug-introducing changes. In: Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering(ASE'06). (2006) 81–90
11. Kim, S., Whitehead, Jr., E.J.: How long did it take to fix bugs? In: Proceedings of the 2006 international workshop on Mining software repositories(MSR'06). (2006) 173–174
12. Mockus, A., Fielding, R.T., Herbsleb, J.D.: Two case studies of open source software development: Apache and mozilla. ACM Transactions on Software Engineering and Methodology **11**(3) (2002) 309–346
13. Śliwerski, J., Zimmermann, T., Zeller, A.: When do changes induce fixes? In: Proceedings of the 2005 international workshop on Mining software repositories (MSR'05). (2005) 1–5
14. Wang, Y., Guo, D., Shi, H.: Measuring the evolution of open source software systems with their communities. SIGSOFT Softw. Eng. Notes **32**(6) (2007) 7
15. Yilmaz, C., Williams, C.: An automated model-based debugging approach. In: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering. (2007) 174–183
16. Carmel, E.: Global software teams: collaborating across borders and time zones. Prentice Hall PTR, Upper Saddle River, NJ, USA (1999)

17. Karolak, D.W.: Global Software Development: Managing Virtual Teams and Environments. IEEE Computer Society Press, Los Alamitos, CA, USA (1999)
18. Herbsleb, J.D., Mockus, A., Finholt, T.A., Grinter, R.E.: An empirical study of global software development: distance and speed. In: Proceedings of the International Conference on Software Engineering. (2001) 81–90
19. Milewski, A.E., Tremaine, M., Egan, R., Zhang, S., Kobler, F., O'Sullivan, P.: Guidelines for effective bridging in global software engineering. In: Proceedings of the International Conference on Global Software Engineering. (2008) 23–32
20. Sangwan, R., Bass, M., Mullick, N., Paulish, D.J., Kazmeier, J.: Global Software Development Handbook (Auerbach Series on Applied Software Engineering Series). Auerbach Publications, Boston, MA, USA (2006)
21. Bird, C., Nagappan, N., Devanbu, P., Gall, H., Murphy, B.: Does distributed development affect software quality? an empirical case study of windows vista. In: ICSE '09: Proceedings of the 2009 IEEE 31st International Conference on Software Engineering, Washington, DC, USA, IEEE Computer Society (2009) 518–528
22. Nguyen, T., Wolf, T., Damian, D.: Global software development and delay: does distance still matter? In: Proceedings of the International Conference on Global Software Engineering. (2008) 45–54