An Analysis Method for Improving a Bug Modification Process in Open Source Software Development

Akinori Ihara Masao Ohira Ken–ichi Matsumoto

Graduate School of Information Science,
Nara Institute of Science and Technology
8916-5, Takayama, Ikoma, Nara, JAPAN 630-0192
tel.+81(743)-72-5318 fax.+81(743)-72-5319
{ akinori-i, masao, matumoto } @ is.naist.jp

ABSTRACT

As open source software products have evolved over time to satisfy a variety of demands from increasing users, they have become large and complex in general. Open source developers often face with challenges in fixing a considerable amount of bugs which are reported into a bug tracking system on a daily basis. As a result, the mean time to resolve bugs has been protracted in these days. In order to reduce the mean time to resolve bugs, managers/leaders of open source projects need to identify and understand the bottleneck of a bug modification process in their own projects. In this paper, we propose an analysis method which represents a bug modification process using a bug tracking system as a state transition diagram and then calculates the amount of time required to transit between states. We have conducted a case study using Firefox and Apache project data to confirm the usefulness of the analysis method. From the results of the case study, we have found that the method helped to reveal that both of the projects took a lot of time to verify results of bug modifications by developers.

Categories and Subject Descriptors

D.2.5 [SOFTWARE ENGINEERING]: Testing and Debugging; D.2.9 [SOFTWARE ENGINEERING]: Management; H.5.3 [INFORMATION INTERFACES AND PRESENTATION]: Group and Organization Interfaces—Collaborative computing, Computer–supported cooperative work, Web–based interaction

General Terms

MANAGEMENT, MEASUREMENT

Keywords

bug tracking system, open source software development, modification process, repository mining, Firefox, Apache

IWPSE-Evol'09, August 24–25, 2009, Amsterdam, The Netherlands.

1. INTRODUCTION

A collective effort by geographically-distributed developers enables us to use high quality and functionality open source software products. As open source software products have evolved over time to satisfy a variety of demands from increasing users, they have become large and complex in general. As a result, open source developers often face with challenges in fixing a considerable amount of bugs which are reported into a bug tracking system [4, 19, 12, 17] on a daily basis.

A bug tracking system is a system for sharing bug information reported by project members including users, knowing the progress of bug modifications, avoiding unmodified bugs and so forth. Although a bug tracking system is designed to help developers collaboratively modify software bugs, the current situation where a lot of bug information is reported impedes efficient bug modification activities. The mean time to resolve bugs has been protracted in these days.

For instance, the mean time to resolve bugs in Linux kernel development was 1.8 years (1.25 median years) [14]. These facts indicate it takes a long time to resolve all reported bugs in large-scale open source software development. Reducing the mean time to resolve bugs is demanded not only from users who want to use open source products safely and comfortably, but also from developers who need to deal with a lot of bugs.

In order to reduce the mean time to resolve bugs, managers and/or leaders of OSS projects need to identify and understand the bottleneck of a bug modification process in their own projects. In this paper, we propose an analysis method which represents a bug modification process using a bug tracking system as a state transition diagram and then calculates the amount of time required to transit between states. Using Firefox and Apache project data, we also conduct a case study to evaluate the usefulness of our analysis method. From the results of the case study, we have confirmed that the method helped to reveal that both of the projects took a lot of time to verify results of bug modifications by developers.

The paper is laid out as follows. Section 2 describes related work and the motivation of our study. Section 3 presents a bug modification process in a common open source project, and then Section 4 proposes an method for analyzing the mean time to resolve bugs based on the modification process. Section 5 reports results of a case study using Apache HTTP Server Project and Mozilla Firefox Project data. Section 6 discusses the results of the case study and

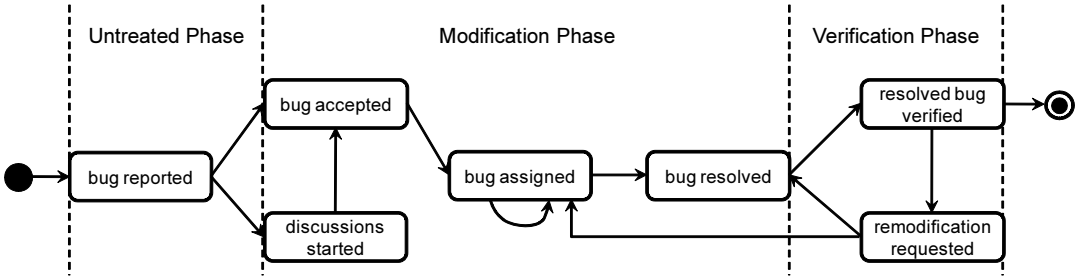

Figure 1: A bug modification process using a bug tracking system

the usefulness of our method. Finally, Section 7 concludes the paper and presents our future work.

2. RELATED WORK

This section introduces past studies on defect analysis in industry software development and bug tracking systems in open source software development.

2.1 Traditional Methods for Defect Analysis

One of major factors of cost overrun and delivery delay in the traditional software engineering is defects which occur during software development. Many analysis methods had been proposed to analyze occurrence factors of defects.

For instance, ODC (Orthogonal Defect Classification) can help developers and testers identify phases and tasks to be improved, by classifying each defect into several defect types and determining whether or not the bias of the number of defects among phases and among tasks [1, 3, 6]. DCA (Defect Causal Analysis) analyzes relationships between factors of defects and the number of defects, by collecting detailed information on each defect through interviews and questionnaires in natural language and then hierarchically organizing defect factors using Fishbone diagrams [5, 13, 16].

Since these studies described above mainly focused on the improvement of software development processes in proprietary software development organizations, the proposed method might not be applied to analyzing bug modification processes in open source projects. In general, open source projects do not have a well-defined development process and planed resources. Different types of analysis methods and/or frameworks are required in order to bug modification processes in open source projects.

2.2 Analysis of Bug Tracking Systems

There are also many studies on bug modification processes with bug tracking systems in open source projects [2, 7, 8, 9, 11, 10, 15, 18, 20, 21].

For instance, Wang et al. proposed several metrics to measure the evolution of open source software [20]. 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 to resolve bugs. In this paper we propose a method to analyze the amount of time to resolve bugs in open source projects at the fine-grained level.

Mockus et al. [15] and Herraiz et al. [8] have reported studies on the mean time to resolve bugs in open source software development. Mockus et al. [15] 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 [15, 8] mainly focused on precise understandings of bug modification processes in open source software development, we are interested in the extraction of the bottleneck in the bug modification process. The aim of our study is to help managers/leaders in open source projects to improve the modification processed in their won projects.

3. BUG MODIFICATION PROCESS

This section describes a bug modification process using a bug tracking system in a common open source project and then define several terms used in the paper.

3.1 Bug Modification Process with a Bug Tracking System

Most open source projects use bug tracking systems to unify management of bugs found and reported by developers and users in their projects. A bug tracking system helps an open source project to know the progress of bug modifications, to avoid leaving unmodified bugs and so forth. Popular bug tracking systems include Bugzilla [4], Mantis [12], RedMine [17], Trac [19][1] and so on.

Figure 1 represents a bug modification process using a bug tracking system. Although a bug modification process using a bug tracking system slightly differs among individual bug tracking systems, it substantially can be represented as a state transition diagram in Figure 1.

Table 1 shows possible states of a bug in the modification process. By calculating the amount of time required to transit between states and the amount of bugs passing through

[1]RedMine and Trac are widely used for project management. Bug-tracking functions are a part of the systems.

Table 1: States of a bug in a modification process with a bug tracking system

state	description
bug reported	Developers and/or users send a report on a found bug into a bug tracking system.
discussions started	Developers start discussions with a message board of a bug tracking system.
bug accepted	Developers check a bug report and then accept a bug.
bug assigned	A bug is assigned to developers.
bug resolved	Developers resolve a bug.
re-modification requested	If a bug fix was not enough, re-modifications are requested to developers.
resolved bug verified	Developers verify whether a bug fix is correct or not.

any two states, we can perform a fine-grained analysis to find out the bottleneck in the modification process.

3.2 Three Phases in a Modification Process

As illustrated in Figure 1, in this paper, we define a bug modification process using a bug tracking system as a process consisting of three different phases: untreated phase, modification phase, and verification phase.

The untreated phase focuses on a sub-process where bugs are reported into a bug tracking system but have not been accepted nor assigned to anyone. According to the study [20], over ten thousand bugs reported into a bug tracking system of the Ubuntu project were not assigned to anyone and left in a state of untreated (no action to modify a bug). So, we consider that it is worth analyzing the untreated phase to precisely reveal delay factors of bug modifications.

The modification phase is a sub-process where bugs are substantially modified. In this phase, a reported bug is accepted to be fixed and then assigned to developers. If the developers finish to modify the bug, the state of the bug transits to "bug resolved". Analyzing this phase would let us know the actual amount of time open source developers spend to modify bugs.

Finally, the verification phase is a sub-process where members in charge of quality assurance verify that modified bugs are correctly resolved. The verification of bugs as well as the bug modification is a heavy-duty task [21]. If a bug modified by developers was not verified, the reported bug would not be recognized as a fixed (closed) bug. The analysis of the verification phase would help managers/leaders in open source communities to determine the proper number of quality assurance members.

Comparing with the past studies [8, 15] on the mean time to resolve bugs in bug tracking systems, the analysis of bug tracking systems based on the division of the bug modification process might provide us new insights on the delay of bug modifications.

4. ANALYSIS METHOD

This section describes an analysis method for identifying the factors which prolong the mean time to resolve bugs in bug tracking systems. The analysis method provides a means to analyze the amount of time from "bug reported" to "bug verified" at the fine grained level and helps to identify the bottleneck in the complicated bug modification process.

4.1 Overview

Using history data of modifications managed by bug tracking systems, the analysis method represents a bug modification process as a state transition diagram and then obtains the amount of time spent for transitions to each state. Representing the bug modification process as a state transition diagram such as Figure 1 allows us to overview the mean time to resolve bugs.

Toward improving the bug modification process and shortening the mean time to resolve bugs, the analysis method can help managers/leaders in open source communities to gain understandings of the bottleneck of the bug modification process and of a precise picture of the process in their own projects.

4.2 Target Data

In this paper the analysis method only targets fixed and closed bugs. Currently unfixed (opened) bugs are not the target of the method. It neither target unaccepted bug reports. Unaccepted bugs may happen when developers do not recognized them as bugs; developers cannot reproduced the same situation as reported by a user; reported phenomena are not due to bugs but specifications: same bugs are already reported and fixed.

A bug tracking system can be used for not only reporting bugs but also requesting new features of software products as a sort of a task management system. Such feature requests registered in a bug tracking system are outside of the scope of our analysis method, because in this paper we are interested in focusing on a bug modification process.

4.3 Procedure

4.3.1 Process definition and data acquisition

A bug modification process must be defined before using the analysis method, because it slightly varies depending on the type of bug tracking systems and management policies in open source projects. For instance, in the Mozilla Firefox project, there are other paths from "bug accepted" to "bug resolved" (e.g., a direct path from "bug accepted" to "bug resolved", a path from "bug assigned" to "bug accepted", and so on).

After defining a bug modification process in a targeted open source project, history data of a bug tracking system is collected. The history data includes when bug states are changed by whom.

4.3.2 Creating state transition diagrams

Using collected history data of a bug tracking system, a state transition diagram is created. The diagram has two roles to understand a bug modification process: a common state transition diagram with state transition probabilities and a diagram focusing on time to transit from one state to another state. The detail of the diagram will be introduced in the case study.

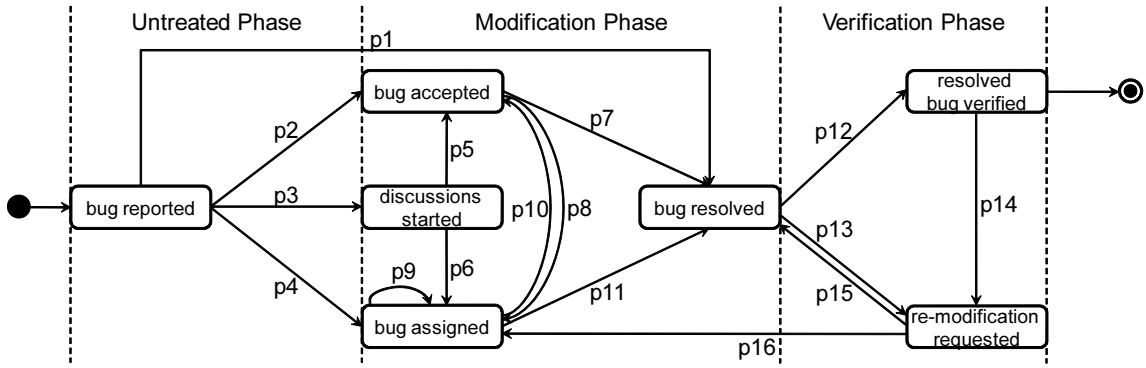

Figure 2: A bug modification process using Bugzilla

Table 2: Statics of Apache and Firefox

	Apache	Firefox
period (from)	2002/03/17	2001/01/01
period (to)	2008/11/30	2008/12/10
fixed and closed bugs [analysis target]	747	2,376
feature requests	545	4,509
currently modifying bugs	2,257	42,729
modified but unverified bugs	671	5,171
unmodified but closed bugs	1,444	8,201
total number of reported bugs	5,664	62,986

4.3.3 *Analysis based on modification phases*

As described in Section 3.2, the analysis method divides a bug modification process into the three phases: untreated phase, modification phase, and verification phase. Calculating and analyzing the amount of time spent in each phase helps managers/leaders in open source projects to identify the longest phase in the bug modification process and then to take measures to improve the current bug modification process in own projects.

4.3.4 *Analysis based on state transition diagrams*

As presented in Figure 1 and Table 1, a bug is modified through several states in the bug modification process. Since previous studies did not reveal elapsed days spent to transit between any two states and the number of bugs passed through any paths, the bottleneck in a bug modification process using a bug tracking system could not be identified.

Our state transition diagrams allow managers/leaders to perform fine-grained analysis of the modification process and to monitor the progress of bug modifications in their own projects. In the future work, we would like to provide a means for monitoring the bug modification progress in real-time.

5. CASE STUDY

This section describes a case study which has been conducted to confirm the usefulness of the analysis method mentioned in Section 4. In the case study, the analysis method was applied to bug modification processes in two open source projects: Apache HTTP Sever project and Mozilla Firefox project.

5.1 Target Projects and Data

In the Apache HTTP Sever project and the Mozilla Firefox project, Bugzilla [4] is used to manage reported bugs. Brief summaries of the two projects are as follows.

- Apache HTTP Server Project
  The Apache HTTP Server project has been developing a web server software product with high functionality and scalability. The product is recognized as high quality open source software and has the biggest market share. The project has been using Bugzilla since 2002. In the case study, history data of Bugzilla in Apache version 1.3, 2.0, and 2.2 had been examined.

- Mozilla Firefox Project
  The Mozilla Firefox project has been developing a web browser product with a rapidly increasing share. The product is very popular due to the extensibility of functions (i.e., add-ons). The project has been using Bugzilla since 2001. In the case study, history data of Bugzilla in Firefox version 1.0, 2.0, and 3.0 had been examined.

Figure 2 shows the bug modification process in the Apache and Firefox projects. We can see different paths of state transitions from Figure 1. The actual modification process in the Apache and Mozilla projects includes additional paths: p1 ("bug reported" to "bug resolved"), p4 ("bug reported" to "bug assigned"), p6 ("discussions started" to " bug assigned"), p7 ("bug accepted" to "bug resolved"), p10 ("bug assigned" to "bug accepted") and P13 ("bug resolved" to "re-modification requested").

Table 2 shows statistics of the target data. The total number of bug reports for the target periods is 5,665 for Apache

Table 3: The mean time to resolve bugs in the Apache project by each phase

	elapsed days in the untreated phase	elapsed days in the modification phase	elapsed days in the verification phase	total elapsed days
average (days)	35.20	82.22	105.00	191.27
median (days)	0.42	15.14	51.45	171.14
standard deviation	68.75	174.28	118.46	197.20
variance	4726.09	30374.94	14032.87	38886.46
maximum (days)	258.72	2153.45	843.08	2295.03
minimum (days)	0.00	0.00	0.00	0.00
number of bugs	86	747	747	747

Table 4: The mean time to resolve bugs in the Firefox project by each phase

	elapsed days in the untreated phase	elapsed days in the modification phase	elapsed days in the verification phase	total elapsed days
average (days)	6.79	57.50	80.03	152.33
median (days)	0.05	9.35	9.88	58.17
standard deviation	32.22	133.40	173.18	236.47
variance	1038.04	17794.76	29992.20	55916.73
maximum (days)	478.96	2333.01	1877.17	2416.00
minimum (days)	0.00	0.00	0.00	0.00
number of bugs	1,631	2,376	2376	2,376

and 62,986 for Firefox. These bug reports include feature requests (545 for Apache and 4,509 for Firefox). Currently modifying bugs (2,257 for Apache and 42,729 for Firefox), unverified bugs (671 for Apache and 5,171 for Firefox) and unmodified but closed bugs (1,444 for Apache and 8,201 for Firefox) are excluded from our analysis, since we cannot know when currently open bugs will be fixed and closed in order to calculate the mean time to resolve them.

In this paper, the analysis method targets data of fixed and closed bugs (747 for Apache and 2,376 for Firefox) to present a precise picture of bug modification processes in the Apache and Firefox projects. Although the target periods of the case study are different between Apache (80 months) and Firefox (105 months), the number of fixed and closed bugs in Firefox is much larger (nearly three times larger) than that in Apache. This would be due to the differences of domains; Apache is a server software product and Firefox is a desktop application.

5.2 Analysis Based on Modification Phases

Table 3 and Table 4 respectively show statistics of bugs in the three modification phases (untreated, modification, and verification phase) and the mean time (the total elapsed days) to resolve bugs in the Apache and Firefox projects.

Note that the statistics in the untreated phase do not include the bug report data of the direct state transition from "bug reported" to "bug resolved", because developers would be likely to resolve bugs before reporting bugs into Bugzilla in case of the direct state transition from "bug reported" to "bug resolved". For in stance, 661 of 747 (88.5%) bugs had the direct state transition from "bug reported" to "bug resolved" in the Apache project. We consider that those data should not be counted in the untreated phase. Instead, we included them in the modification phase.

In the both projects, the minimum elapsed days in each phase were 0.00. This can happen when developers find a bug, fix it, and then report it into Bugzilla. In contrast, the maximum elapsed days were over 200 days (e.g., over 6 years in the modification phase of the Firefox project). Since these bugs might be outliers due to some reasons, it seems to be adequate to use the median days to discuss the mean time to resolve bugs. We can see that elapsed all the median days in Firefox were shorter than Apache.

Figure 3 and Figure 4 show distributions of the number of reported bugs in each modification phase ((a), (b), (c)) and in the total of the three phase ((d)). In Figure 3and Figure 4, the X-axis and Y-axis respectively mean elapsed days to resolve bugs and the number of reported bugs.

In the untreated phase, 51 of 86 (59.3%) bugs for Apache and 491 of 1,631 (30.1%) bugs for Firefox spent within a day, while 21 of 86 (24.4%) bugs for Apache and 61 of 1,631 (5.4%) bugs for Firefox spent over 90 days. In the modification phase, 172 of 747 (23.0%) bugs for Apache and 542 of 2,376 (22.8%) bugs for Firefox spent within a day, while 199 of 747 (26.6%) bugs for Apache and 412 of 2,376 (17.3%) bugs for Firefox spent spent over 90 days. In the verification phase, 180 of 747 (24.1%) bugs for Apache and 483 of 2,376 (20.3%) bugs for Firefox spent within a day, while 314 of 747 (42.0%) bugs for Apache and 504 of 2,376 (21.2%) bugs for Firefox spent over 90 days. In total, over the half of Apache bugs spent over 90 days to be fixed (closed), while the half of Firefox bugs were fixed within 90 days.

As a result of our analysis using the proposed method, we have found that elapsed days in the untreated phase were shorter than that in the modification and verification phases in both of the Apache and Firefox projects. We can conclude that elapsed days in the modification and verification phase have an effect on the total prolongation of the mean time to resolve bugs in Apache and Mozilla.

Although these results are not surprising results but expected, we also have found that there were bugs elapsed over three months in the untreated phase (i.e., 24.4% of Apache's bugs and 5.4% of Firefox's bugs in the untreated phase) from

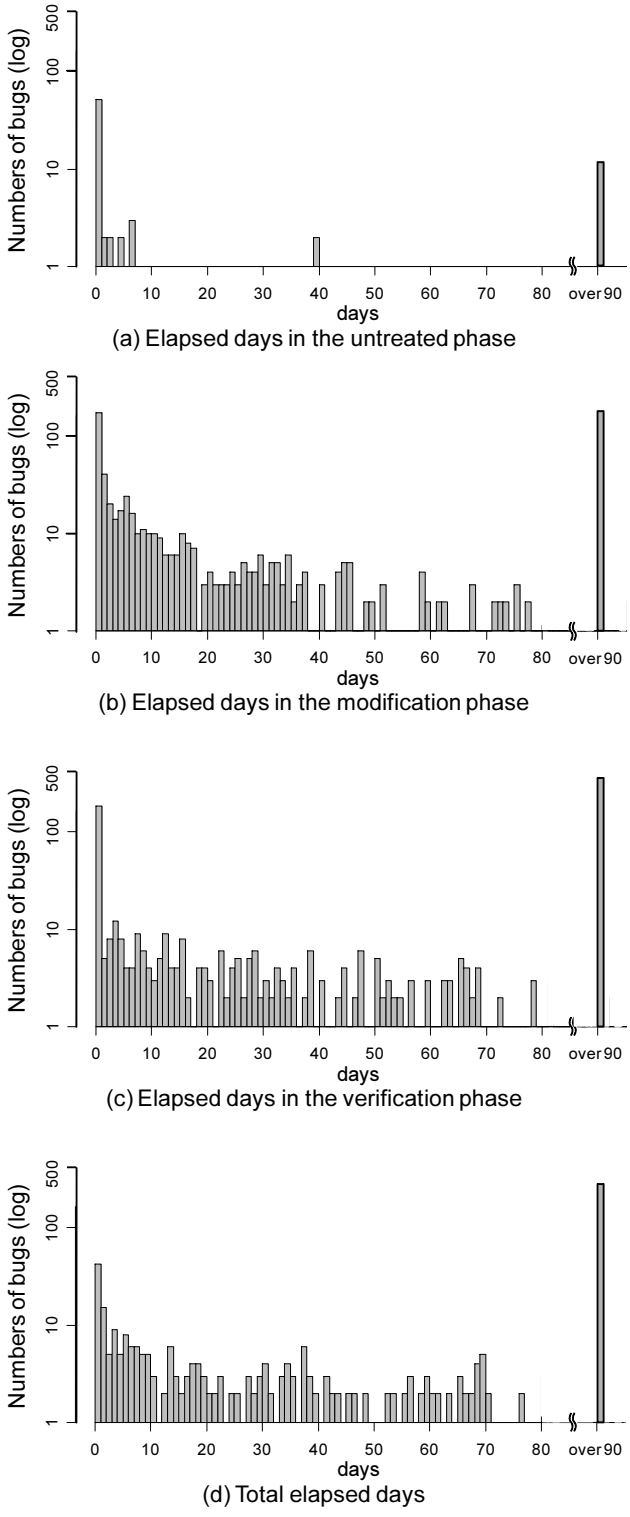

Figure 3: The distribution of the number of bugs and elapsed days of the Apache project in (a) the untreated phase, (b) the modification phase, (c) the verification phase, and (d) the total phase. Bugs with over 90 elapsed days are summed up on the right side of the histograms.

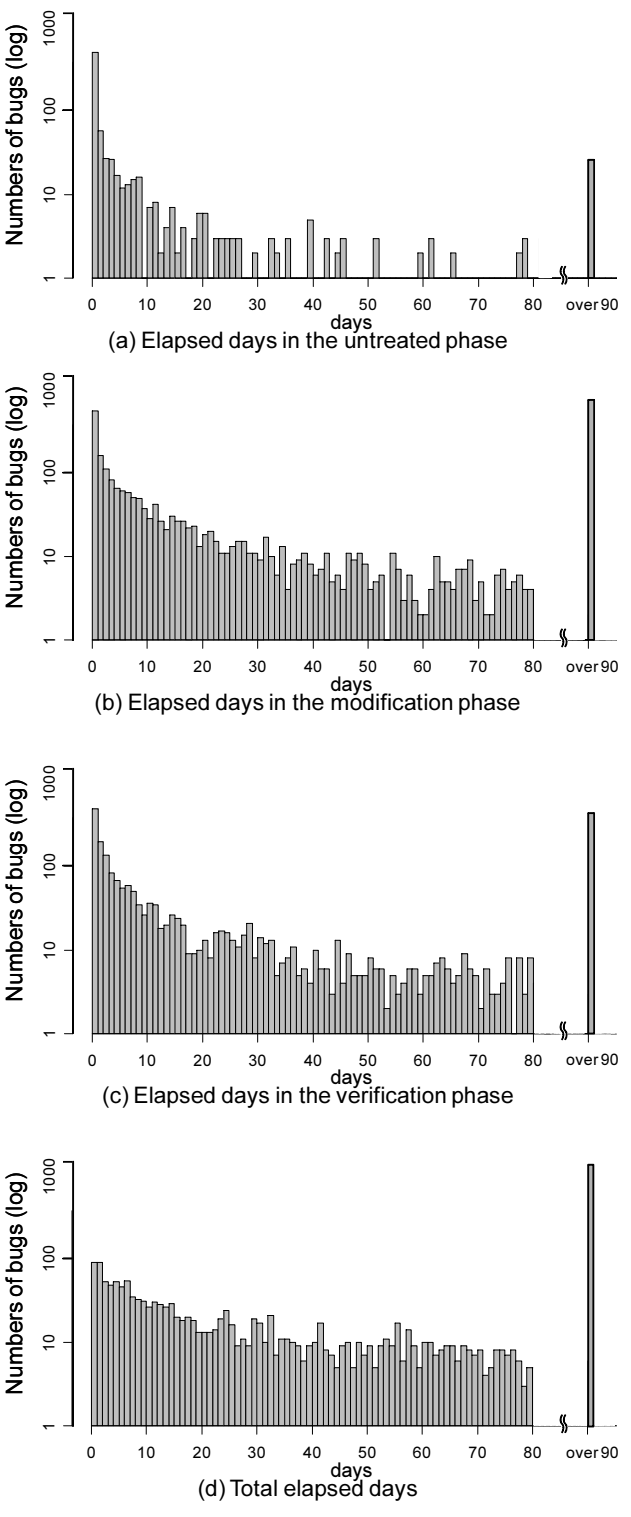

Figure 4: The distribution of the number of bugs and elapsed days of the Firefox project in (a) the untreated phase, (b) the modification phase, (c) the verification phase, and (d) the total phase. Bugs with over 90 elapsed days are summed up on the right side of the histograms.

Figure 3 and Figure 4. These bugs might not largely influence on the total prolongation of the mean time to resolve bugs, but it might not be acceptable for the projects to ignore them.

5.3 Analysis Based on State Transition Diagrams

Figure 5 and Figure 6 show the bug modification process in Apache and Firefox respectively. In the figures, the values written above the lines (arcs) show the amount of time (days) spent to transit between states. and also shows the bug modification process in Apache and Firefox respectively, but, in these figures, the values written above the lines (arcs) mean the number of the bugs transited between states and their probabilities (i.e., state transition probabilities).

Note that the values are calculated by simply counting how many bugs passed through each arch. The analysis method did not take into account existences of looped paths and self-loops. We need to consider this issue in the future.

5.3.1 *Analyzing the time spent to transit between states*

Using the analysis method, two transitions consume most of the time in both Apache and Firefox projects:

1 The transition from "bug resolved" to "resolved bug verified" spent the largest amount of time to transit.

2 The transition from "bug accepted" to "bug resolved" spent the second largest amount of time to transit.

However, the third largest amount of time to transit was different between the Apache and Firefox projects. In the Apache project, the third largest amount of time to transit was the transition from "bug assigned" to "bug resolved". In contrast, in the Firefox project, the third largest amount of time to transit was the transition from "bug accepted" to "bug assigned". We also found that in the both projects, the transition from "bug assigned" to "bug resolved" spent shorter time to transit than the transition from "bug accepted" to "bug resolved".

5.3.2 *Analyzing the bug modification sub-processes*

As illustrated in Figure 5 and Figure 6, the method could visualize how many bugs passed through which path, by dividing the entire bug modification process into fine grained sub processes. The figures showed that the bugs without accepted nor assigned (the direct path from "bug reported" to "bug resolved") were 661 (88.5%) in the Apache project and 745 (31.4%) in the Firefox project.

In the Firefox project, over the half of reported bugs (1,501 of 2,376 (63.2%)) transited to "discussions started" and then moved to "bug accepted" (442 of 1,501 bugs) or "bug assigned" (1,079 of 1,501 bugs). The Firefox project had more "re-assigned bugs" (393 of 2,079 assigned bugs (18.9%)) than that (3 of 96 assigned bugs (3.1%))of the Apache project. We can also see that the Firefox project had more bugs (1,422 bugs) resolved after assigning them to developers than that (90 bugs) of the Apache project.

6. DISCUSSIONS

Based on the results of our case study, this section discusses the usefulness of the analysis method.

6.1 Analysis Based on Modification Phases

Since the past studies on the mean time to resolve bugs [8, 15, 20] only analyzed the total amount of time from "bug reported" to "bug verified", they could not identify which phase has the modification delay. By dividing the bug modification process into the three semantic phases, our analysis method allowed us to figure out that elapsed days in the modification and verification phase had an effect on the prolongation of the mean time to resolve bugs in the Apache and Mozilla projects.

We could also find that there were bugs elapsed over three months in the untreated phase (i.e., 24.4% of Apache's bugs and 5.4% of Firefox's bugs in the untreated phase). From these results, we consider that the analysis method would help managers/leaders of open source projects to gain understandings of the bottleneck of the bug modification process and of a precise picture of the process in their own projects.

6.2 Analysis Based on State Transition Diagrams

In order to reduce the mean time to resolve bugs, managers/leaders of open source projects firstly need to identify and understand the bottleneck of a bug modification process in their own projects. As a result of applying the analysis method to the bug modification processes in the Apache and Firefox projects, we have confirmed that the analysis method could identify the bottlenecks of the modification processes in the two projects: from "bug resolved" to "bug verified" and from "bug accepted" to "bug resolved".

Although elapsed days from "bug resolved" to "bug verified" were the longest transitions between any two states, we could not confirm the reason (e.g., bugs were closely-inspected in the period? or quality assurance members were late to get round to the verification?). We need to further analyze those reasons to provide a support for reducing the mean time to resolve bugs.

We could also find that there were the three major patterns of elapsed days from "bug reported" to "bug resolved" as follows.

(a) from "bug reported" to "bug resolved"

(b) from "bug accepted" to "bug resolved"

(c) from "bug assigned" to "bug resolved"

We expected that the pattern (a) should be the fastest path because developers could modify bugs before reporting bugs. However, the pattern (b) and (c) were faster than the pattern (a) in both of the Apache and Firefox projects. This indicates that bug assignments are critically important for reducing the mean time to resolve bugs. In this way, we could obtain the appropriate path for efficient modifications by using the analysis method.

We also found that there were different bottlenecks between Apache and Firefox. While the third longest state transition in Apache project was from "bug assigned" to "bug resolved", the third longest state transition in the Firefox project was from "bug accepted to "bug assigned". In this fashion, the analysis method helped us to find bottlenecks of bug modification processes according to characteristics of individual projects.

From these results described above, we can conclude that the analysis method is useful for managers/leaders of open

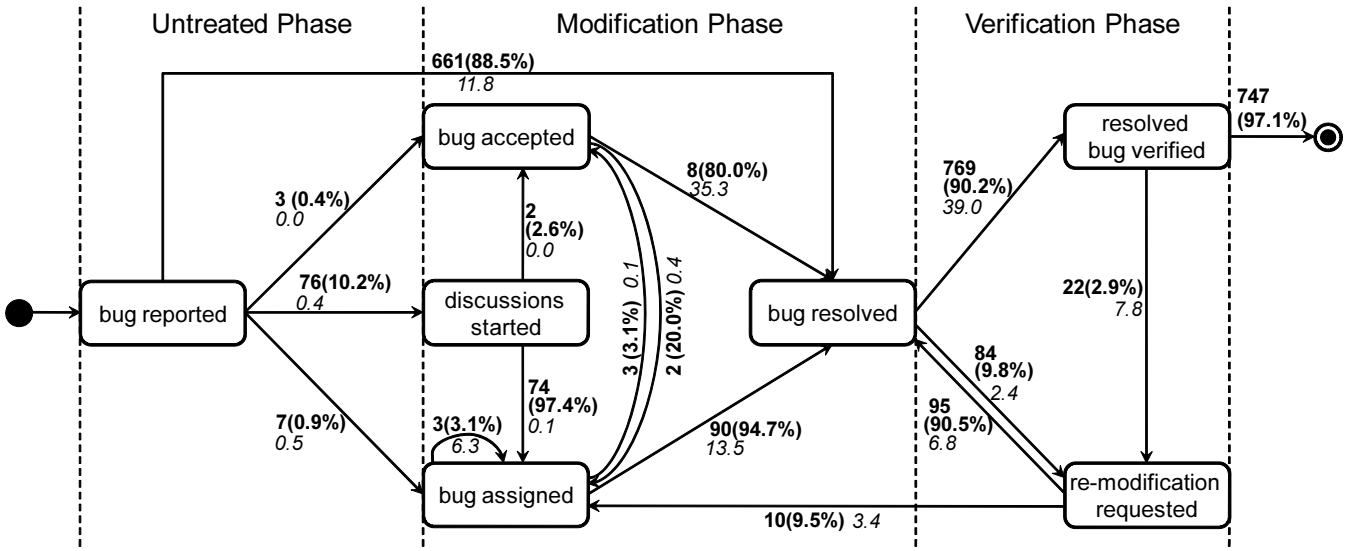

Figure 5: The bug modification process in the Apache project. The bold numbers on the arcs are the number of bugs and state transition probabilities. The italic numbers are elapsed days for a transition between two states.

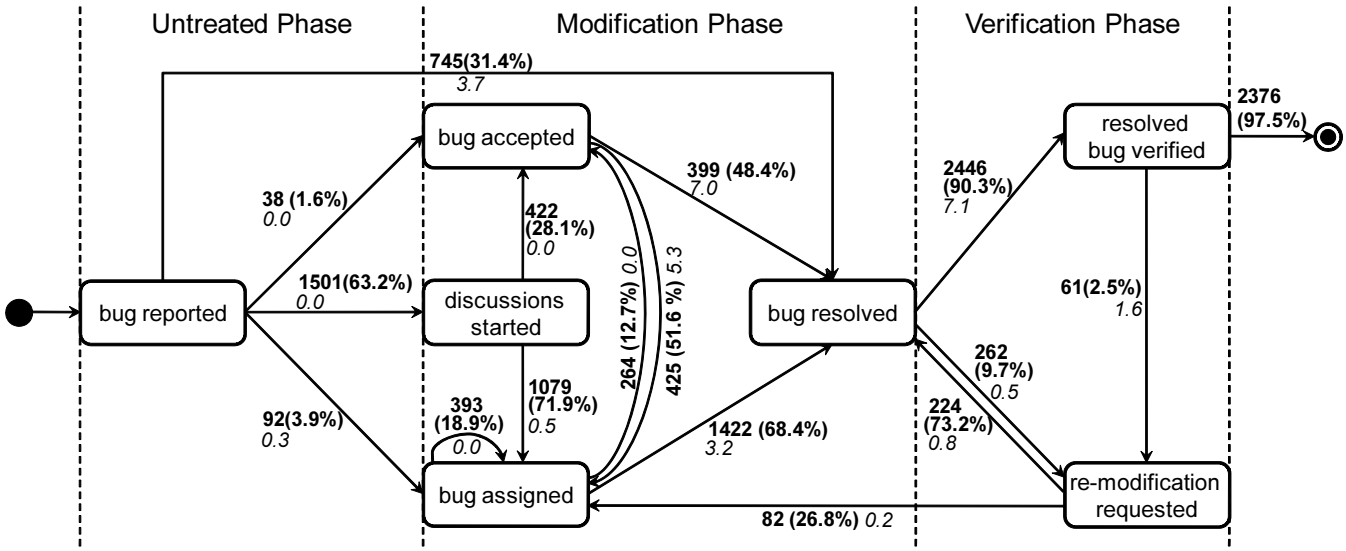

Figure 6: The bug modification process in the Firefox project. The bold numbers on the arcs are the number of bugs and state transition probabilities. The italic numbers are elapsed days for a transition between two states.

source projects to gain understandings of the bottleneck of the bug modification process and of a precise picture of the process in their own projects, toward improving the bug modification process and shortening the mean time to resolve bugs.

6.3 Threats to Validity

In this paper, we defined the start of the modification phase as the time of "bug accepted" or "discussion started". But the actual start time of modifications by developers can be earlier than the time when developers report bugs. In the future, we need to precisely define the start time of the modifications by using CVS commit logs.

The current analysis method does not take account of looped paths and self loops in the state transitions. In Figure 5 and Figure 6, represented transition probabilities might be derived based on calculations of looped paths and self loops. We have to consider how many times a bug transits to the same state.

7. CONCLUSION AND FUTURE WORK

In this paper, we proposed an analysis method which represents a bug modification process using a bug tracking system as a state transition diagram and then calculates the amount of time required to transit between states. Using

Firefox and Apache project data, we also conducted a case study to evaluate the usefulness of our analysis method.

From a case study using Firefox and Apache project data, we have confirmed the usefulness of the method as follows.

- The method helped to identify the phase to be reduced by dividing the bug modification process into the three phases.
- The method could find out the bottleneck of a bug modification process by representing it as a state transit diagram consisting of fine grained sub processes.
- The method could extract the difference of modification processes between Firefox and Apache.
- As empirical findings, the method could identify that verifying resolved bugs were the most time-consuming task in both of the Apache and Firefox projects .

The results of our case study suggested that the analysis method could provide managers/leaders of OSS projects with useful information as demonstrated in Section 5, toward improving the bug modification process and shortening the mean time to resolve bugs.

In this paper, however, we did not show the relationships between the states of bugs and the attributes of bugs (e.g., severity, priority, and so on). In the future, the analysis method should be improved to provide useful insights on such the relationships.

8. ACKNOWLEDGEMENT

We appreciate the anonymous reviewers giving insightful comments and helpful suggestions. We would like to thank Asa Dotzler of the Mozilla project for giving us a lot of useful comments and advises on the case study. We also would like to thank Mizuki Yamamoto for helping us analyze open source protect data.

This research is being conducted as a part of the Next Generation IT Program and Grant-in-aid for Young Scientists (B), 20700028, 2009 by the Ministry of Education, Culture, Sports, Science and Technology, Japan.

9. REFERENCES

[1] K. A. Bassin, T. Kratschmer, and P. Santhanam. Evaluating software development objectively. IEEE Software, 15(6):66–74, 1998.

[2] 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(FSE'08), pages 308–318, 2008.

[3] I. Bhandari, M. Halliday, E. Tarver, D. Brown, J. Chaar, and R. Chillarege. A case study of software process improvement during development. IEEE Transactions on Software Engineering, 19(12):1157–1170, 1993.

[4] Bugzilla. http://www.bugzilla.org/.

[5] D. N. Card. Learning from our mistakes with defect causal analysis. IEEE Software, 15(1):56–63, 1998.

[6] R. Chillarege, I. Bhandari, J. Chaar, M. Halliday, D. Moebus, B. Ray, and M.-Y. Wong. Orthogonal defect classification-a concept for in-process measurements. IEEE Transactions on Software Engineering, 18(11):943–956, 1992.

[7] 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), pages 131–142, 2000.

[8] I. Herraiz, D. M. German, J. M. Gonzalez-Barahona, and G. Robles. Towards a simplification of the bug report form in eclipse. In Proceedings of the 2008 international working conference on Mining software repositories (MSR'08), pages 145–148, 2008.

[9] S. Kim, K. Pan, and E. E. J. Whitehead, Jr. Memories of bug fixes. In Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering(FSE'06), pages 35–45, 2006.

[10] S. Kim and E. J. Whitehead, Jr. How long did it take to fix bugs? In Proceedings of the 2006 international workshop on Mining software repositories(MSR'06), pages 173–174, 2006.

[11] S. Kim, T. Zimmermann, K. Pan, and E. J. J. Whitehead. Automatic identification of bug-introducing changes. In Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering(ASE'06), pages 81–90, 2006.

[12] Mantis. http://www.mantisbt.org/.

[13] R. G. Mays, C. L. Jones, G. J. Holloway, and D. P. Studinski. Experiences with defect prevention. IBM Systems Journal, 29(1):4–32, 1990.

[14] A. Michail and T. Xie. Helping users avoid bugs in gui applications. In Proceedings of the 27th international conference on Software engineering(ICSE'05), pages 107–116, 2005.

[15] A. Mockus, R. T. Fielding, and J. D. Herbsleb. Two case studies of open source software development: Apache and mozilla. ACM Transactions on Software Engineering and Methodology, 11(3):309–346, 2002.

[16] T. Nakajo and H. Kume. A case history analysis of software error cause-effect relationships. IEEE Transactions on Software Engineering, 17(8):830–838, 1991.

[17] RedMine. http://www.redmine.org/.

[18] J. Śliwerski, T. Zimmermann, and A. Zeller. When do changes induce fixes? In Proceedings of the 2005 international workshop on Mining software repositories (MSR'05), pages 1–5, 2005.

[19] Trac. http://trac.edgewall.org/.

[20] Y. Wang, D. Guo, and H. Shi. Measuring the evolution of open source software systems with their communities. SIGSOFT Softw. Eng. Notes, 32(6):7, 2007.

[21] 6 C. Yilmaz and C. Williams. An automated model-based debugging approach. In Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, pages 174–183, 2007.