2012 19th Asia-Pacific Software Engineering Conference

SATA Workshop

An Investigation on Software Bug-Fix Prediction for Open Source Software Projects -A Case Study on the Eclipse Project -

Akinori Ihara*, Yasutaka Kamei†, Akito Monden*, Masao Ohira‡, Jacky Wai Keung§,
Naoyasu Ubayashi† and Ken-ichi Matsumoto*
* Nara Institute of Science and Technology, 8916-5 Takayama, Ikoma, Japan
Email: (akinori-i, akito-m, matumoto)@is.naist.jp
† Kyushu University, 744 Motooka, Nishi-ku, Fukuoka, Japan
Email: (kamei, ubayashi)@ait.kyushu-u.ac.jp
‡ Wakayama University, 930, Sakaedani, Wakayama, Japan
Email: masao@sys.wakayama-u.ac.jp
§ Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong
Email: Jacky.Keung@comp.polyu.edu.hk

***Abstract*—Open source software projects (OSS) receive a large number of bug reports from various contributors and developers alike, where many planned to be fixed by OSS developers. Given the next release cycle information, OSS users can be more effective and flexible in planning and to fix the bugs that are not to be fixed in the next release. It is therefore vital for OSS users to learn which bugs the OSS developers will fix, unfortunately such information may not be readily available, nor there is a prediction framework exists to serve such an important purpose. In this study, we would like to answer the question "Will this bug be fixed by the next release?", this is addressed by building a bug fixing prediction model based on the characteristics of a bug-related metric and by incorporating the progress of bug fixing measures such as status, period and developer metrics to provide aggregated information for the OSS users. The proposed model calculates the deviance of each variable to analyze the most important metrics, and it has been experimented using a case study with Eclipse platform. Result shows a bug fixing prediction model using both base metrics and state metrics provide significantly better performance in precision (139%) and recall (114%) than the standard model using only base metrics.**

I. Introduction

Large Open Source Software (OSS) developers do not fix all bugs which reported to the project. Because, the number of bug reports is too many for the project. For example, Mozilla Firefox project developers do not fix 84.4% bugs (29,399 bugs / 34,778 bugs) [1]

Even if a company developer contacts to OSS developers to ask to fix a bug, they will not quickly answer response it except in the case of be claimed the same problem by many users [13]. Therefore, when company developers detect the bug in OSS, they need to know whether the OSS developers will fix the bug by the next release. Even if the OSS developers will not fix the bug, the company developers need to consider fixing the bug themselves.

In this study, we reveal that the important metrics to know whether OSS developers will fix the bug by the next release from the characteristics metrics of the bug and the progress metrics of the bug-fix, and then build a bug-fix prediction mode based on the metrics. Using large OSS project (Eclipse platform), we answer two research questions.

RQ1: What metrics are the most important to determine in the characteristics of the bug and the progress of the bug-fix?

RQ2: How accurate is the bug-fix prediction model built?

Many researchers have proposed models to predict how long to take developers to fix bugs [6][10][11][21]. They use the characteristics metrics (e.g. component, priority) of the bug to build the model. On the other hand, we try to build the model whenever company developers can predict how long OSS developers need to fix bugs. Therefore, we use not only the characteristics metrics of the bug, but also the progress metrics (fixer, change of bug info, period from the reported time) of the bug fixing.

In our experiment, the model predicts the bug will be fixed in 3 month from the predicted date[2]. And the model is built based on the characteristics metrics of the bug and the 3 kinds of progress metrics (status, period, developer) of the bug-fix. Status metrics mean the state of the bug in predicted date, and the period to the predicted date from the reported date. Period metrics mean the reported date, and the period from changing the bug report to the predicted date. Developer metrics mean reporter's name, assignor's name, and assignee's name.

This paper is laid out as follows. Section 2 describes

[1]We checked it at Feb 2011.

[2]OSS project will announce at least 3 months before release date.

related work and the motivation of our study. Section 3 presents the bugs which were fixed in OSS project. Section 4 provides the design of our experiment, and Section 5 gives the results. Section 6 discusses the results of the experiment and developer activity after becoming committers. Finally, section 7 concludes the paper and presents our future work.

II. Related work

Software developers sometimes take a long time to fix bugs. [3][4][12]. Therefore, many researchers have proposed a method to predict time to fix bugs for making software test plan or software release plan [6][7][10][11][21].

Hewett et al. [10] presented method to predict time to fix bugs based on the characteristics metrics of the bug. Also, Cathrin et al. [21] presented method to predict it based on the time to fix the similar bugs. However, they target only fixed bugs (about 8% bugs in Mozilla Firefox), because they focus on predicting how long developers take to fix bugs. On the other hand, we target on finding all the bugs (both fixed bugs and not fixed bugs).

Philip et al. [7] presented method to predict whether developers should respond to the bug reported to a project. Especially, Hooimeijer et al. [11] focus on OSS projects, because OSS developers decide to fix or not to fix in their project. Some of the bugs (UNFIXED BUGS) were not responded by OSS developers because of some reasons (e.g. They cannot catching the cause of the bugs.) If OSS developers know that the bug is UNFIXED BUGS, they do not need to take time to respond. However, company developers need to fix the bug by themselves. So, it is not enough for company developers to predict whether the bug is the treated bugs or not.

Existing research uses only the feature of the bug (base metrics) to predict time to fix bugs. However, company developers sometimes want to know when the bugs will be fixed by OSS developers. Then, the progress of the bug such as status metrics will contribute to build the bug-fix prediction model. Also, existing research helps for developers which predict time to fix the bug, and they target a just reported bug. On the other hand, we help users which predict bugs fixed by OSS developers and we target both a just reported bug and a fixing bug.

III. Bug Fixing Process in an OSS project

In this section, we present the bug fixing process in an OSS project and the differences between FIXED BUGS (which are fixed by OSS developers) and NOT FIXED BUGS (which are not fixed by OSS developers).

A. Bug Fixing Process

Most open Source projects use bug tracking system to unify management of bugs which were 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 fix. Popular bug tracking systems include Bugzilla[3], Trac[4], RedMine[5] and so on. When developers and users report a bug, they note a basic information (e.g. product, version, priority, and severity), and a bug detail information (which is the way to find the bug). After that, bug tracking system manages the progress of the bug fix, and the discussion between developers and users.

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

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

Figure 1 shows the bug fixing process.

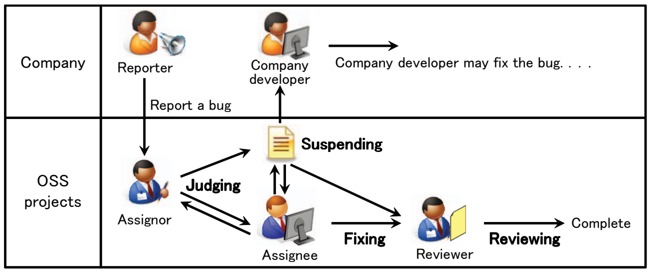

Fig. 1. Common bug modification process.

Judging. Assignors judge whether OSS developers should fix the bug which was reported to OSS projects.

Fixing. Assignors assign a bug fix task to the assignee. And the assignee fixes the bug.

Verifying. Reviewers verify the bug which was fixed by the assignee. Or the reviewers verify whether the suspended bug should be fixed.

Suspending. Assignor or assignee do not fix the bug for the following reason. (1) They cannot replicate the bug. (2) If they fix the bug or add new function, OSS performance will be worse. (3) They cannot contact to the reporter.

B. Bug Fixing Pattern

Developers and users report a lot of bugs to OSS projects. There are 3 types of bugs (FIXED BUGS, UNFIXED BUGS, FIXING BUGS) in an OSS project.

FIXED BUGS means the bugs which are finished verifying after fixing them. In figure 1, the bug report status changes to **Complete** after **Fixing** and **Verifying**.

UNFIXED BUGS (NOT FIXED BUGS) means the bugs which are judged as not to fix by OSS developers, because of the cause of the bugs are unknown or being not possible to replicate the issue. In Figure 1, the bug report status transits to **Complete** after **Suspending** and **Verifying**.

FIXING BUGS (NOT FIXED BUGS) means the bugs which are fixing or suspending. In figure 1, the bug report status has not transited to **Complete** yet.

In this paper, we built a model to predict the bugs which will be **FIXED BUGS** by the next release.

IV. Research Question

To predict the bugs which will be FIXED BUGS based on the characteristics metrics of bugs and the kinds of progress metrics of the bug fix, we have two research questions. In this section, we present the motivation of the research questions

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

and the method related to the experimentation of the Rsesearch Questions.

RQ1: What metrics are the most important to determine in the characteristics of the bug and the progress of the bug fix?

Motivation. Some of bugs will be fixed by OSS developers. On the other hand, the remaining bugs are not fixed by OSS developers. If company developers detect the bugs in their product using OSS, they have to fix those bugs. However, it is difficult for the company developers to know the bugs which are fixed by OSS developers. Of course, company developers will not understand the bug fix plan in the OSS project. Then, which information in bug report should company developer check to know the bugs which are fixed by OSS developers? For RQ1, we analyze the most important metrics from the base metrics and the 3 kinds of progress metrics (status, period, developer) to predict the bug which will be fixed by the next release.

Approach. We calculate the deviance of each variable to analyze the most important metrics. The deviance means the changed amount of the likelihood (the rate to be output the correct result by a model), when new variable is added to the model. Therefore, the bigger deviance, the more the metrics contribute to the model. In this study, we summarize every deviance in each metrics, and then we divide the deviance of the alternate hypothesis (the model built based on some variables) into the deviance of the null hypothesis (the model based on intercept) to understand how much the metrics contributes to the model.

RQ2: How accurate is the bug-fix prediction model built?

Motivation. Company developers have to identify whether the bug will be fixed by OSS developers based on the information noted bug reports. For RQ2, we built a bug-fix prediction model using the base metrics and the 3 kinds of progress metrics, and evaluated the model.

Approach. We built 4 bug-fix prediction models. Then we compare them. One is the model using only the base metrics. Another is the model using each progress metrics. Another is the model using all metrics (the base metrics and the progress metrics). The other is the model using one metrics which is chosen from the base, status, period, and developer metrics randomly.

To predict bugs which will be fixed by the next release, we use a logistic regression model [2][15][19]. Because, our data set include the nominal scale data and non-normal distribution data. The objective variable is whether or not a bug will be fixed by the next release (i.e., FIXED BUG or NOT-FIXED BUG). The model identify **FIXED BUG**, when the output (continuous value: 0-1) of the model is over a threshold (0.5). However, when the correlation of two variables is more than 0.8, we will remove the one to avoid multicollinearity.

The evaluation metrics are precision, recall and F1 value [8][14]. The precision means the ratio of the number of bugs which actually were fixed by the next release to the number of predicted bugs which were fixed by the next release. The recall means the ratio of the number of bugs which became actually were fixed by the next release to the number of bugs which never been fixed by the next release. F1-value is a combined value of recall and precision as follows.

$$F1{-}value = \frac{2 \times Recall \times Precision}{Recall + Precision}$$

The evaluation method is 10-fold cross-validation [5]. First, the dataset are divided into 10 equal parts. The model is built on 9/10 of the dataset (i.e. the training data), and then evaluated on the remaining 1/10 of the dataset (i.e. the independent testing data). This experiment is repeated 1000 times. We calculate the median of the evaluation result (precision, recall, and F1-value) in each experiment, and then we compare the result in each model.

V. Metrics for building the bug-fix prediction model

We use 41 variables to build a bug-fix prediction model. Table I shows the variables. In the existing research, they have used only base metrics to build the model. On the other hand, we also use the 3 kinds of the progress metrics (status, period, and developer).

A. Base metrics and Progress metrics

Base metrics. Base metrics indicates a general metrics which most of existing researches use to predict the bug fix time [10][11]. This metrics includes the characteristics of bugs (e.g. Component, OS, Hardware, and Priority), the number of comments (NumComment), and the number of attachments (e.g. patch, screenshot).

Status metrics. Status metrics indicates the state on the predicted date, and the status history change from reported date to predicted date. This metrics includes the bug state on the predicted date (CurrentStatus), the presence or absence of assignee (ChAssignee), the number of transition in each status (NumModifying, NumSuspending), and the period from the status changed date to the predicted date (PeriodOnStatus). When the status of bug is **Judging** or **Suspending**, OSS developers will take a long time to assign the bug and to start the bug fixing [1][18] .

Period Metrics. Period metrics include the reported year (ReportedYear), the reported month (RepotedMonth), the period from base metrics (e.g. component, priority) the period from the changed date to the predicted date (LastComponetnTime, LastPriorityTime). If the base metrics have never changed, the period is from the reported date to the predicted date. Reporter sometimes reports the wrong content. Then OSS developers will take a long time to fix bugs, because they have to look for the appropriate assignee again.

Developer Metrics Developer Metrics includes the reporter (Reporter), the assignee (Assignee), the assignor (Assignor), and the number of changed assignee (NumAssignee). In OSS

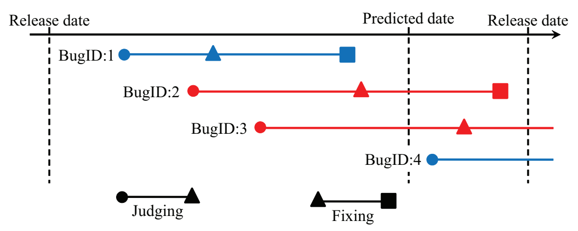

Fig. 2. Target bug reports.

project, a reporter sometimes fixes a bug by himself. In this case, the bug will be fixed soon. Because, reporter has known the cause of the bug [13].

B. Nominal scale in our experiment

We use some nominal scale variables such as component name, reporter name. In our experiment, the nominal scale is expanded dummy variable to build the bug-fix prediction model. However, when Component and Reporter are expanded as dummy variable, the number of variables is too many. If we use many dummy variables, the prediction accuracy of the model will go down. Because it is difficult to build the appropriate model [17][20]. Therefore, we try that top 3 of the highest appearance frequency are made dummy variables and the others one are grouped together as " others " . For example, Component includes com-A, com-B, com-C, com-D and com-E in order. In this case, com-A, com-B and com-C are made dummy variables, and com-D, and com-E are changed " others " .

VI. Experiment

In this section, to understand effective metrics to predict the bugs will be fixed by the next release, we build the bug-fix prediction model using the base metrics and the progress metrics.

A. Experimental overview

We target bugs which developers and users reported after last release. Using the bugs, we predict whether they are fixed by the next release in 3 months. Figure 2 shows the target bugs in our experiment. Also, we focus on the bugs which have not been fixed by the predicted date (e.g. BugID:2 and 3 in figure 2). Bug ID 2 is **FIXED BUGS**, and also Bug ID 3 is **NOT FIXED BUGS**. The model identifies bugs which will be fixed by the next release.

B. Target Project

Using Eclipse platform project, we find the effective metrics to predict the bugs which will be fixed by the next release. Table II shows the number of bugs in the target projects. In this section, we target the project which (1) use Bugzilla as the bug tracking system (2) release on a regular basis (3)have enough bugs to compare the results in each project.

TABLE II
Summary of the target data.

	Number of Bugs
All bugs	2,276
FIXED BUGS	183
NOT FIXED BUGS	2,093

TABLE III
Deviance explained in each category metrics.

	Deviance
base	27.19
status	11.50
period	2.04
developer	3.27
all	44.00

C. Experimental results

RQ1: What metrics are the most important in the characteristics of the bug and the progress of the bug-fix?

We calculated the deviance of each metrics to find the most important metrics for the model. Table III shows the deviance of each metrics and the deviance of whole metrics. Also, table IV shows top 10 variables which have the most highest deviance in each project.

In the analysis of the metrics deviance, we found that the base metrics is the most important metrics to build the model. Also, status metrics is the second most important metrics to build the model.

In the analysis of the variable deviance, we found that one of the most important variables is whether the bug's priority is low or general (Priority_low, Priority_general). OSS developers have a tendency to put off the low priority bugs [9][16]. Also, another one of the most important variables is about bug's status such as PeriodOnStatus_Modifying_0-30 and CurrentState_Judging. For example, if the bug's status transits to new status early, OSS developer will fix the bug soon. I will discuss the period in this discussion section.

RQ2: How accurate is the bug-fix prediction model built?

Table V shows the evaluation result (precision, recall, and F1-value) of the bug-fix prediction model and the model built using one of the metrics randomly. And, we compare the evaluation result of the bug-fix prediction model which has the highest F-1 value to the evaluation model based on only base metrics to the bug-fix prediction model built using one of the metrics randomly. As a result, the F-1 value of the model built using multiple metrics including the base metrics is better than the model built using in each metrics. And, the model based on the base metrics and the status metrics is the highest F1-value in both projects. We found that precision and recall are about 31% and about 71% in the Eclipse platform project. The evaluation result of the model based on Base metrics and Status metrics improves than the result of the model based on only the base metrics. Precision improves about 39%, and recall improves about 29%. Also, the evaluation result of the model based on Base metrics and Status metrics improves than

TABLE I
SUMMARY OF THE METRICS.

Metrics	Variable name	Scale	Summary
base	NumDescriptionWord	interval	the number of words in bug report's description
	NumAttachment	interval	the number of attached files in a bug report
	Keywords	nominal	presence or absence of the bug's keyword
	Component	nominal	the name of component which bug relates to
	Priority	nominal	level of priority (high, normal, low)
	Severity	nominal	level of severity (high, normal, low)
	Os	nominal	the name of operating system which reporter uses
	Hardware	nominal	the type of hardware which reporter uses (e.g. x86, PowerPC)
	Milestone	nominal	the name of milestone which the bug relate to
	NumComment	interval	the number of comment between developers and users
	Cc	interval	the number of persons which have received the update of the bug report
	Blocks	interval	the number of bugs which block the bug fix
	Dependson	interval	the number of bugs which depends on the bug
status	NumModifying	interval	the number of transiting to Modifying
	NumSuspending	interval	the number of transiting to Suspending
	ChAssignee	nominal	presence or absence of the bug's assignor
	CurrentStatus	nominal	the state of bug on the predicted date
	PeriodOnStatus	interval	the period from the date changed bug's state to the predicted date
period	ReportedYear	nominal	reported year
	ReportedMonth	nominal	reported month
	LastAttachmentTime	interval	the period from date attached new files to the predicted date
	LastKeywordsTime	interval	the period from date changed the the keyword to the predicted date
	LastComponentTime	interval	the period from date changed the target component to the predicted date
	LastPriorityTime	interval	the period from date changed the priority to the predicted date
	LastSeverityTime	interval	the period from date changed the severity to the predicted date
	LastOsTime	interval	the period from date changed the target OS to the predicted date
	LastHardwareTime	interval	the period from date changed target hardware to the predicted date
	LastMilestoneTime	interval	the period from date changed the milestone to the predicted date
	LastCommentTime	interval	the period from date submitted new comment to the predicted date
	LastCcTime	interval	the period from date added/deleted email address in CC list to the predicted date
	LastBlocksTime	interval	the period from date added/deleted a bug blocking the bug to the predicted date
	LastDependsonTime	interval	the period from date added/deleted an assignee to the predicted date
	LastAssigneeTime	interval	the period from date added/deleted a depending on the bug to the predicted date
	RepNowTime	interval	the period from the reported date to the predicted date ()
developer	Reporter	nominal	reporter's email address
	Assignee	nominal	assignee's email address
	Assignor	nominal	assignor's email address
	Reporter_Assignor	nominal	reporter and assignor is same or not
	Assignor_Assignee	nominal	assignor and assignee is same or not
	Reporter_Assignee	nominal	reporter and assignee is same or not
	NumAssignee	interval	the number of the changed assignee

TABLE IV
DEVIANCE OF TOP 10 VARIABLES.

No.	Variables
1	Milestone_3.5M7
2	**CurrentState_Judging**
3	**CurrentState_Modifying**
4	Milestone_3.5
5	RepNowTime
6	Priority_general
7	Component_IDE
8	**Modifying_0-30**
9	Reporter_dev03
10	***Priority_low***

the result of the model based on the one metrics which is chosen from the base, status, period, and developer metrics randomly. Precision improves from about 29%, and recall improves from about 25%.

In our experiment, we found that the precision was not so high (Eclipse platform: about 31%). Because, the number of **NOT FIXED BUGS** is much more than the number of **FIXED BUGS**. If output of the logistic (threshold) is more than 0.5, the precision improves. For example, when the threshold is 0.8, precision and recall are about 43% and about 59% in the Eclipse platform project. However, the threshold should not be too high, because recall will go down.

VII. DISCUSSION

A. High possibility of the fixed bug

As a result of RQ2, we found that the bug-fix prediction model based on the base metrics and the status metrics has higher accuracy than the bug-fix prediction model based on only base metrics. Also, we found that status metrics is important metrics to predict the bug will be fixed by the next release. Using odds ratio (OR), we analyze whether we can know the bug will be fixed by the next release, if we use the status metrics such as the status of bugs and the period from the date changed status to the predicted date. Odds ratio means scale to compare the likelihood of a consequence between the 2 groups. Odds ratio is calculated according to the following

TABLE V
Bug-fix prediction result.

		Precision	Recall	F1-value
Predicive model	base	0.23	0.62	0.33
	status	0.24	0.66	0.35
	period	0.18	0.52	0.27
	developer	0.31	0.49	0.29
	base+status	**0.31**	**0.71**	**0.43**
	base+period	0.26	0.69	0.38
	base+developer	0.24	0.66	0.35
	all metrics	0.25	0.78	0.38
Random		0.24	0.57	0.33
Percentage of improvement from prediction model (only base metrics) (%)*		1.39	1.14	1.31
Percentage of improvement from random prediction (%)**		1.29	1.25	1.30

* This indicates how accuracy the evaluation result of the bug-fix prediction model based on the base metrics and the status metrics improve more than the bug-fix prediction model based on only the base metrics.
** This indicates how accuracy the evaluation result of the bug-fix prediction model based on the base metrics and the status metrics improve more than the andom prediction.

equation.

$$OR = \frac{p/(1-p)}{q/(1-q)}$$

p means the rate of bugs with any status and any period from the date changed status to the predicted date. q means the rate of bugs with the other status and the other period from the date changed status to the predicted date. For example, 138 **FIXED BUGS** and 1,690 **NOT FIXED BUGS** with **Judging** have passed in 1 month from the date changed the status in the predicted date in Eclipse project. In this case, p is about 0.08 ($\frac{138}{138+1690} \approx 0.08$). 45 FIXED BUGS and 403 NOT FIXED BUGS with **Judging** have passed for more than 1 month from the date changed the status in the predicted date in Eclipse project. In this case, q is about 0.10 ($\frac{45}{45+403} \approx 0.11$). Then, Odds ratio of the bug whose status is **Judging** and which has passed in 1 month is 0.73.

Table VI show the result of Odds ratio. 3rd line means p value, and 4th line means q, 5th line means Odds ratio. When the Odds ratio is more than 1, the bugs with any status and any period from the date changed status are more likely to be fixed by the next release than the other bugs. For example, when target bug is with **Judging** have passed in 1 month (0-30) from the date changed the status, the Odds ratio is 0.73. As a result, the bug is less likely to be fixed by the next release, because the score is less than 1.

Table VI shows the Odds ratio of the bugs with any status and any period from the date changed status. As a result, the longer the bugs with **Judging** have passed from the date changed to the status, the more the bugs is likely to be fixed by the next release in Eclipse platform project..

When the target bugs' status is **Fixing**, almost bugs' odd ratio is more than 1. Therefore the bugs are more likely to be fixed by the next release than the other bugs. Especially, the bugs with **Fixing** which have passed in 1 month from changing the status is more likely to be fixed. We also found it in RQ1. On the other hand, bugs with **Suspending** are more likely to be fixed by the next release than the bugs with **Judging** or **Fixing** in Eclipse platform project. However, bugs with **Suspending** is less likely to be fixed.

In brief, the bugs with **Judging** or **Fixing** are likely to be fixed by the next release. On the other hand, the longer the bugs with **Suspending** have passed from the date changed to the status, the less the bugs is likely to be fixed. The status and the period of the bug will indicate whether bug will be fixed by the next release.

B. *Ineffective metrics for the bug-fix prediction model*

As a result of RQ2, we found that the period metrics is ineffective for the bug-fix prediction model. It may be for this reason that we excluded the many variables which correlate highly with the other variables to avoid multicollinearity.

Also, we found that the developer metrics is ineffective for the bug-fix prediction model. It may be for this reason that the number of the target developers (reporter, assignor, and assignee) is a few. We focused on the top 3 developers in each variable. And the other developers are grouped together as "others". Table VII shows the top 10 developers which have reported/assigned/fixed a lot of bugs. The table shows the developers name, the number of bug reports which the developers have concerned, and the number of the bugs which have finished the bug-fix by the next release. We found that the most of bugs concerned by some developers have not been fixed. For example, most of bugs concerned by dev02, dev11, dev18, and dev19 have not been fixed by the next release in Eclipse platform project. If the developer metrics has been top 3 developers and the others such as our experiment, the developer metrics does not so contribute to improve the accuracy of the bug-fix prediction model. Because the fixed bugs depends on developers. In the future, we will propose a method to extract the developers which contribute the bugs fix.

TABLE VI
ODDS RATIO FOR THE FIXED BUGS OR THE NOT FIXED BUGS IN EACH BUG STATE IN ECLIPSE PLATFORM PROJECT

Status	Judging				Fixing				Suspending			
Period	0-30	30-90	90-180	180-	0-30	30-90	90-180	180-	0-30	30-90	90-180	180-
p	0.08	0.10	0.36	0.57	0.42	0.42	0.11	0.30	0.07	0.07	0.04	0.00
q	0.11	0.07	0.07	0.08	0.08	0.08	0.09	0.06	0.09	0.09	0.10	0.09
Odds ratio	0.73	**1.49**	**5.13**	**6.92**	**5.09**	**5.18**	**1.31**	**4.85**	0.81	0.77	0.40	0.00

TABLE VII
THE NUMBER OF BUGS WHICH A DEVELOPER CONSIDERED.

	Reporter				Assignor				Assignee			
	Name	After changing Name	The number of submission bugs	The number of fixed bugs	Name	After changing name	The number of submission bugs	The number of fixed bugs	Name	After changing name	The number of submission bugs	The number of fixed bugs
1	dev01	dev01	43	14	—	other	740	46	—	other	740	46
2	dev02	dev02	39	1	dev11	dev11	219	0	dev06	dev06	126	23
3	dev03	dev03	38	12	dev12	dev12	215	8	dev02	dev02	119	0
4	dev04	other	34	2	dev06	dev06	194	17	dev16	dev16	108	2
5	dev05	other	33	8	dev13	other	184	5	dev17	other	95	3
6	dev06	other	31	5	dev14	other	122	2	dev18	other	84	0
7	dev07	other	31	8	dev04	other	60	3	dev12	other	69	16
8	dev08	other	26	1	dev03	other	59	0	dev19	other	61	0
9	dev09	other	24	6	dev15	other	57	4	dev13	other	61	13
10	dev10	other	22	4	dev16	other	50	5	dev11	other	57	0

C. Threats to Validity

Existing research has tried the bug-fix time prediction used only base metrics. We built the bug-fix prediction model using the 4 metrics (41 variables) from only the bug tracking system, because, it is difficult for the company developer to extract the other metrics (e.g. Cyclomatic complexity, Code Dependencies) from the other repository (e.g. Version control system, Mailing list).

We tried the experiment using Eclipse platform project. In both projects, we got the same results (RQ1: we found that the base metrics and the status metrics are the most important metrics to build the model. RQ2: the F-1 value of the model built using multiple metrics including the base metrics is better than the model built using in each metrics.). Even if we target the other large OSS projects for our experiment, we will get the same results. However, if we target a project which has different feature (e.g. release frequency, small project), we will get the different results.

In our experiment, we set that the predicted date is before 3 month of the release. Because, many large OSS project release major version more than once a year. If company developers set the long period (more than for 3 month) from the predicted date to the next release, an OSS project may change the release date. Then, they have to build the model again. On the other hand, if they set the short period (less than for 3 month) from the predicted date to the next release, there are few FIXED BUGS. It is difficult to predict the bugs.

VIII. CONCLUSIONS AND FUTURE WORK

In this study, we built a bug-fix prediction model based on the base metrics and 3 kinds of the progress metrics to identify bugs which is fixed by OSS developers by the next release. Using data from the Eclipse platform project, we found:

RQ1: we found that the base metrics and the status metrics are the most important metrics to build the model.

RQ2: the F-1 value of the model built using multiple metrics including the base metrics is better than the model built using in each metrics.

In this study, we found that our bug-fix prediction model can identify bugs that are fixed by OSS developers better than existing approach manually. In fact, company developers depend on a status of bug and some comments that are indicated the interest of bug for OSS developers. However, even if company developers see the status and the comments of only a bug, it is difficult for them to know the release plan and the development plan in OSS project. Although OSS projects were described to establish a transparent development process, it is actually difficult for company developer to know them. This study will be first step to establish a transparent development process in OSS project.

In the future, we would like to enhance the bug-fix prediction model to use new metrics based on the content of the bug-fix. Then, the prediction accuracy of FIXED BUGS will improve.

ACKNOWLEDGMENT

This research is being conducted as a part of the Next Generation IT Program and Grant-in-aid for Young Scientists (B), 22700033, 2010 and (Start-up), 23800044, 2011 by the Ministry of Education, Culture, Sports, Science and Technology, Japan.

REFERENCES

[1] J. Anvik, L. Hiew, and G. C. Murphy. Who should fix this bug? In *Proceedings of the 28th International Conference on Software Engineering (ICSE'06)*, pages 361–370, 2006.

[2] V. R. Basili, L. C. Briand, and W. L. Melo. A validation of object-oriented design metrics as quality indicators. In *IEEE Transactions on Software Engineering*, volume 22, pages 751–761, 1996.

[3] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim. Duplicate bug reports considered harmful... really? In *Proceedings of the 24th International Conference on Software Maintenance (ICSM'08)*, pages 337 – 345, 2008.

[4] G. Canfora and L. Cerulo. Supporting change request assignment in open source development. In *Proceedings of the 21st Symposium on Applied Computing (SAC'06)*, pages 1767–1772, 2006.

[5] B. Efron. Estimating the error rate of a precision rule: improvements on cross-validation. In *Journal of the American Statical Association*, volume 78, pages 316–331, 1983.

[6] E. Giger, M. Pinzger, and H. Gall. Predicting the fix time of bugs. In *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering (RSSE'10)*, pages 52–56. ACM, 2010.

[7] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy. Characterizing and predicting which bugs get fixed: an empirical study of microsoft windows. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE'10)*, pages 495–504, 2010.

[8] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. In *ACM Transaction Information Systems*, volume 22, pages 5–53, 2004.

[9] 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 5th International Working Conference on Mining Software Repositories (MSR'08)*, pages 145–148.

[10] R. Hewett and P. Kijsanayothin. On modeling software defect repair time. In *Empirical Software Engineering*, volume 14, pages 165–186. Kluwer Academic Publishers, 2009.

[11] P. Hooimeijer and W. Weimer. Modeling bug report quality. In *Proceedings of the 22nd International Conference on Automated Software Engineering (ASE'07)*, pages 34–43, 2007.

[12] A. Ihara, M. Ohira, and K. Matsumoto. An analysis method for improving a bug modification process in open source software development. In *Proceedings of the joint International and annual ERCIM Workshops on Principles of Software Evolution and Software Evolution Workshops (IWPSE-Evol'09)*, pages 135–144, 2009.

[13] A. Ihara, M. Ohira, and K. Matsumoto. Differences of time between modification and re-modification: An analysis of a bug tracking system. In *Proceedings of the 3rd International Workshop on Knowledge Collaboration in Software Development (KCSD'09)*, pages 17–22, 2009.

[14] S. Kim, E. J. Whitehead, Jr., and Y. Zhang. Classifying software changes: Clean or buggy? *IEEE Transactions Software Engineering*, 34:181–196, March 2008.

[15] G. Liang, L. Wu, Q. Wu, Q. Wang, T. Xie, and H. Mei. Automatic construction of an effective training set for prioritizing static analysis warnings. In *Proceedings of the 25th International Conference on Automated Software Engineering (ASE'10)*, pages 93–102, 2010.

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

[17] I. Myrtveit, E. Stensrud, and U. H. Olsson. Analyzing data sets with missing data: An empirical evaluation of imputation methods and likelihood-based methods. *IEEE Transactions on Software Engineering*, 27:999–1013, 2001.

[18] S. Rastkar, G. C. Murphy, and G. Murray. Summarizing software artifacts: a case study of bug reports. In *Proceedings of the 32nd International Conference on Software Engineering (ICSE'10)*, pages 505–514, 2010.

[19] Y. Takagi, O. Mizuno, and T. Kikuno. An empirical approach to characterizing risky software projects based on logistic regression analysis. In *Empirical Software Engineering*, volume 10, pages 495–515, 2005.

[20] H. B. K. Tan, Y. Zhao, and H. Zhang. Conceptual data model-based software size estimation for information systems. In *ACM Transaction on Software Engineering and Methodology (TOSTEM)*, volume 19, pages 1–37, October 2009.

[21] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller. How long will it take to fix this bug? In *Proceedings of the 4th International Workshop on Mining Software Repositories (MSR'07)*, pages 1–8, 2007.