Stage-oriented Analysis on Factors Impacting Bug Fixing Time

Hong Wu\textsuperscript{1,2}, Junjie Wang\textsuperscript{1}, Qing Wang\textsuperscript{1,2}, Lin Shi\textsuperscript{1}, Feng Yuan\textsuperscript{1,4}
\textsuperscript{1}Laboratory for Internet Software Technologies, Institute of Software, Chinese Academy of Sciences, Beijing, China
\textsuperscript{2}University of Chinese Academy of Sciences, Beijing, China
\textsuperscript{3}State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
\textsuperscript{4}Institute of Software Application Technology, Guangzhou & Chinese Academy of Sciences, Guangzhou, China
\{wuHong, wangjunjie, wq, shilin\}@itechs.iscas.ac.cn, yf@gz.iscas.ac.cn

Abstract—The timely fixing of bugs is important to ensure software quality. In Open Source Software (OSS) development, behaviors of stakeholders impact the bug fixing process, especially the different stages respectively. However, most of the existing studies on impact factors of bug fixing time usually treat bug fixing process as a whole, while neglecting the particularity at its different stages. Ignoring the detail of different stages cannot let us understand why the fixing time is longer or shorter. In this paper, we aimed at investigating whether the factors have different impacts on the time of different stages and the whole process. Three stages of the whole fixing process were formalized, and twenty-four factors were defined and extracted from three aspects: bug reports, their associated source code and code changes. An empirical study based on two OSS projects, Eclipse JDT Core and Linux Kernel, was conducted for the investigation. The results of our study provide a very positive validation that the influence of factors on bug fixing time is stage related, rather than for the whole process. Our results can help developers better understand influences of factors on the bug fixing process, and thus provide opportunities to improve their process effectively.

Keywords-OSS maintenance; bug fixing time; stage-oriented analysis; empirical study

I. INTRODUCTION

Fixing bugs is an inevitable and time-consuming activity in the software development process. It is estimated that 80\% of the total cost of a software system is spent on fixing bugs [3, 13]. Therefore, it is crucial to investigate the impact factors of bug fixing time, so as to effectively manage the fixing process.

In open source software communities, anyone who has interests in maintaining a software can participate in the bug fixing process, e.g., by posting comments to discuss the cause of a bug, or by taking an assignment of bug fixing, and etc. Behaviors of stakeholder usually change during the whole fixing process, and many stakeholders might only participate at certain stages of the whole process. In this case, stakeholders’ behaviors impact the bug fixing process in an uncertain manner, especially at different stages of the process respectively. Therefore, the self-governing of developers in OSS results in the correlation between stakeholders’ effort and the time of the whole bug fixing process may no longer be formed.

Prior studies have investigated the factors impacting bug fixing time, and proposed techniques to predict the time to fix a bug [1, 4, 7, 8, 11, 17, 19]. Most of them studied the time of the bug fixing process as a whole (i.e., from a bug was reported until it was resolved), while neglecting the particularity at its different stages. However, some studies pointed out that factors with the same value can usually result in different fixing time [12]. This may derive from the inherent nature of the bug fixing process, i.e., the whole process consists of several individual stages, and different stages focus on different sub-activities of the whole process with different stakeholders involved.

Moreover, our observations on two large open source projects (i.e., Eclipse JDT Core and Linux Kernel) show that, along with different stakeholders’ behaviors, the value of many factors are frequently changed during the whole fixing process. For instance, the assignee of bug #13939 in Eclipse JDT Core was changed 7 times. The summary of bug #86231 in Linux Kernel was modified 5 times. Due to the change in the value of these factors, their influence on individual stage might be different from that on the whole process. For example, the location of a bug has been considered as an impact factor of bug fixing time in prior study [8]. However, we found that, in Linux Kernel, the product category of a bug report has significant influence on the bug assignment stage and the bug fixing stage, rather than on the whole process. The above examples motivate this study.

In this paper, we presented a stage-oriented analysis of factors impacting bug fixing time. Three major stages of the bug fixing process were formalized, which are the bug assignment stage, the bug fixing stage and the bug verification stage. Twenty-four factors were defined and extracted from three aspects: bug report, the associated source code and code changes. We conducted an empirical study with three research questions as follows.

RQ1: Do the factors have different correlations with the time of different stages?

RQ2: Do the factors have different correlations between the time of individual stages and the time of the whole process?

RQ3: Which factors have the highest correlation with the time of each stage?

To answer the above three questions, we conducted the empirical study on two large open source projects (i.e., Eclipse JDT Core and Linux Kernel). The results revealed that factors have different influence among individual stage as well as the whole bug fixing process. For instance, the number of
comments and times of re-assignment have higher correlation with the time of the assignment stage than the time of the whole process. We believe our results can help developers better understand the influence of factors on the time of the bug fixing process, and illustrate the necessity of analyzing the influence of factors on bug fixing time by stages instead of the whole process.

Our study is different from prior work in that: we performed a stage-oriented analysis on the factors impacting bug fixing time, taking the change in the value of the factors into account. We believe these findings can provide new viewpoints and important references for efficient bug management, and provide an opportunity of improving current approaches of bug fixing time prediction for OSS.

The remainder of the paper is organized as follows. The setup of our study is described in Section II, and the results on two open source projects to answer our research questions are reported in Section III. The threats to validity are listed in Section IV. After summarizing the related work in Section V, we conclude our work in Section VI.

II. STUDY SETUP

In this section, we present the detailed design of our empirical study.

A. Subject Projects

The subject projects in our study are Eclipse JDT Core (Eclipse for short) and Linux Kernel (Linux for short). We choose these two projects for three reasons: (1) they are highly active projects and have been widely used in practice and prior studies; (2) they are from different domains (Integrated Development Environment vs. Operating System) and written in the two most famous and commonly used program languages (Java and C); (3) their development processes are well-managed with high-quality bug reports by Bugzilla and source code by Git.

B. Formalization of Bug Fixing Process

We first introduced four timepoints, and then defined three stages of the bug fixing process. Figure 1 visually presents the four timepoints and three stages in a timeline.

Bug Reporting Timepoint ($T_R$) is the timestamp when a bug is reported to Bugzilla by a user or developer.

Bug Assignment Timepoint ($T_A$) is the timestamp when a bug is assigned to the appropriate developer through Bugzilla. If reassignment occurred, we denote $T_A$ as the timestamp when a bug is assigned to the developer who fixes the bug. Similar with previous work [17], changing assignee back to the default one (‘xxx-inbox’ in Eclipse or ‘product-component’ in Linux) is not considered as a bug assignment in our study.

Bug Fixing Timepoint ($T_F$) is the timestamp when the commits for fixing the bug are submitted to Git. If multiple commits are linked to one bug report, we use the submission time of the last commit as $T_F$.

Bug Verification Timepoint ($T_V$) is the timestamp when the status of a bug is marked as VERIFIED. We treat the time when a bug is marked as CLOSED or RESOLVED as $T_V$ in the case of the official VERIFIED status is missed out.

Based on these four timepoints, we defined the time of the whole bug fixing process as the interval between $T_R$ and $T_V$, marked as $BLT$ (standing for Bug Life Time). Moreover, we divided the bug fixing process into three main stages as follows.

**Bug Assignment Stage ($S_A$)** is the stage for understanding a bug report and assigning it to an appropriate developer for the fix. The time of $S_A$ is defined as the interval between $T_R$ and $T_A$, which is computed as $I_A = T_A - T_R$.

**Bug Fixing Stage ($S_F$)** is the stage for the developer to fix the assigned bug by modifying the source code files. The time of $S_F$ is defined as the interval between $T_A$ and $T_F$, which is computed as $I_F = T_F - T_A$.

**Bug Verification Stage ($S_V$)** is the stage for reviewers to verify the developers’ resolution on the fix of assigned bugs. The time of $S_V$ is defined as the interval between $T_F$ and $T_V$, which is computed as $I_V = T_V - T_F$.

![Figure 1. The stages of the bug fixing process and related timepoints](image)

C. Definition of Factors

We defined 24 factors, which might influence the bug fixing time, from three aspects: 14 factors related to 8 attributes of bug reports, 3 factors related to the complexity of a bug fixing task measured by the source code, and 7 factors related to the effort required to fix a bug measured by the code changes. TABLE I shows the description of all defined factors. In particular, the scale of $ProdCat$ and $CompCat$ is nominal, the scale of $PriLevel$ and $SevLevel$ is ordinal, and the scale of the rest twenty factors is ratio.

D. Data Collection and Filtering

We collected data from bug reports, the associated source code and code changes to conduct our experiment, because we believe all of them contain the information that can influence bug fixing time for OSS. As we mentioned in Section II.A, for the two subject projects, bug reports were obtained from Bugzilla\(^1\), while source code and code change were obtained from Git\(^1,4\), respectively. The process of data collection and filtering is elaborated as follows.

**Step 1: Retrieve commit logs and bug reports.** We used `git log` to retrieve all commit logs from Git. For the collection of bug reports, we first obtained a set of bug IDs by using Bugzilla’s search engine. In this study, we only collected IDs of fixed bug reports with the following search criteria: (1) the field of Status changed to RESOLVED, VERIFIED or CLOSED before Dec. 31, 2015; (2) The Resolution is marked as FIXED (in Eclipse) or CODE_FIX (in Linux). After removing

\(^1\) https://bugs.eclipse.org
\(^2\) https://bugzilla.kernel.org
\(^3\) git://git.eclipse.org/gitroot/jdt/eclipse.jdt.core.git
\(^4\) git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git
the duplicated IDs in the above search results, we got a set of fixed bug IDs. Then we downloaded the webpage of the bug report associated with each bug ID. In this way, we obtained a set of bug reports.

**Step 2: Link commit logs to bug reports to obtain a preliminary dataset.** To establish the linkage between bug reports and its corresponding fixing commit, we first identify the bug IDs appearing in the commit logs by two patterns: (1) for Eclipse, bug IDs are included in the subject of commit logs; (2) for Linux, bug IDs are appeared in the content of commit logs with the form of the bug’s URL in Bugzilla. If a bug ID was identified in the commit log, we compared it with the set of bug reports obtained in Step 1. If matched, the commit was considered as a bug-fixing commit dedicated to the corresponding bug report. Thus we included a pair of bug report and commit logs as a data item in the preliminary dataset.

### Table I. SUMMARY OF DEFINED FACTORS

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Factor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug Reports</td>
<td>ProdCat</td>
<td>The product category of a bug</td>
</tr>
<tr>
<td></td>
<td>CompCat</td>
<td>The component category of a bug</td>
</tr>
<tr>
<td></td>
<td>PriLevel</td>
<td>The priority level of a bug</td>
</tr>
<tr>
<td></td>
<td>SevLevel</td>
<td>The severity level of a bug</td>
</tr>
<tr>
<td></td>
<td>LenSum</td>
<td>The length of the summary of a bug report in terms of English words</td>
</tr>
<tr>
<td></td>
<td>LenDesc</td>
<td>The length of the description of a bug report in terms of English words</td>
</tr>
<tr>
<td></td>
<td>NumCom</td>
<td>The length of comments for the bug</td>
</tr>
<tr>
<td></td>
<td>LenCom</td>
<td>The length of total comments of a bug report in terms of English words</td>
</tr>
<tr>
<td></td>
<td>CHSum</td>
<td>The number of changes for the summary content of a bug</td>
</tr>
<tr>
<td></td>
<td>CHProd</td>
<td>The number of changes for the product category of a bug</td>
</tr>
<tr>
<td></td>
<td>CHComp</td>
<td>The number of changes for the component category of a bug</td>
</tr>
<tr>
<td></td>
<td>CHPri</td>
<td>The number of changes for the priority level of a bug</td>
</tr>
<tr>
<td></td>
<td>CHSev</td>
<td>The number of changes for the severity level of a bug</td>
</tr>
<tr>
<td></td>
<td>CHAssi</td>
<td>The number of changes for the assignee of a bug</td>
</tr>
<tr>
<td>Source Code</td>
<td>SLOC</td>
<td>The number of source lines of code of all changed files</td>
</tr>
<tr>
<td></td>
<td>NumMethod</td>
<td>The number of methods of all changed files</td>
</tr>
<tr>
<td></td>
<td>Method_CC</td>
<td>The cyclomatic complexity of methods of all changed files</td>
</tr>
<tr>
<td>Code Changes</td>
<td>NumCFile</td>
<td>The number of all changed files</td>
</tr>
<tr>
<td></td>
<td>AddSLOC_M</td>
<td>The number of added source lines within methods</td>
</tr>
<tr>
<td></td>
<td>AddSLOC_F</td>
<td>The number of added SLOC for fields</td>
</tr>
<tr>
<td></td>
<td>AddLOC_C</td>
<td>The number of added lines of changed comments</td>
</tr>
<tr>
<td></td>
<td>DelSLOC</td>
<td>The number of deleted SLOC for changed code</td>
</tr>
<tr>
<td></td>
<td>NumCMethd</td>
<td>The number of all changed methods</td>
</tr>
<tr>
<td></td>
<td>CMethod_CC</td>
<td>The cyclomatic complexity of all changed methods</td>
</tr>
</tbody>
</table>

**Step 3: Filter invalid data items to obtain a filtered dataset.** We filtered out certain invalid data items from the preliminary dataset, and thus a filtered dataset was obtained. The following exclusive criteria were used for filtering.

*Data items that contain bug report whose severity is enhancement:* According to the regulation of Bugzilla, this kind of bug reports is actually the feature request. We filtered them out from the preliminary dataset, because they are out of the scope of our study.

*Data items that contain bug report which was reopened:* We found about 12% data items in both subject projects contained reopened bug reports. Such data items often involved a more complex fixing process, which is quite different from the majority of the data items. Therefore, we filtered out these data items for drawing more general conclusions.

*Data items that contain commits which have no source code changes:* We filtered out the data items which contain commits with configuration files, pictures and etc., because the factors of source code and code changes cannot be extracted from such data items.

*Data items with negative time of the stage of bug fixing process:* About 10% data items in Eclipse and 20% in Linux have negative value of $I_3$ or $I_5$. This may be caused by several reasons. For instance, the developer, who is both the bug triager and the bug fixer, starts to fix a bug and submit commits before he assigns the bug report to himself (e.g., bug #50781 in Linux), or the developer changes the bug report status prior to submitting the fixing commits (e.g., bug #140879 in Eclipse). We filtered out those data to ensure the validity of our outcome.

**Step 4: Retrieve source code and code change files to obtain the final dataset.** For each data item in the filtered dataset, based on its commit logs, we used `git show` to get the detailed code changes of each commit. Moreover, for each commit, we used `git checkout` to get the source code files with the post-fix version and change history logs of those files. Then we get the commit ID in the pre-fix version of each changed source code file from its change history log, and we used `git checkout` again to get the source code files in the pre-fix version. In our study, we defined the pre-fix version of source code file as the first previous version that has diffs with the post-fix version. We used the current version (i.e., post-fix version) for a new added source code file, because there is no pre-fix version for new added source code files in Git.

Finally, we obtained 5493 data items, including 4297 bug reports linked with 4984 commits from Eclipse, and 1196 bug reports linked with 1272 commits from Linux. In particular, for each commit, we obtained its detailed change log, a set of changed source code files in the pre-fix version and those in the post-fix version. Due to the page limitation, we present the detailed statistics of the data set in our project webpage5, and also make the data set available.

**E. Computation of Factor Value**

For factors related with bug reports, we extracted their values from the downloaded webpages. In particular, we parsed the modification history of the bug report, with each modification record (MR) including who, when, removed or added what. We divided MRs into $S_1$, $S_2$ and $S_3$ according to its timestamp as well as the timestamp of each stage illustrated in Section II.B, and then we got the value of each factor in each

5 http://itech.iscas.ac.cn/cn/membersHomepage/wuhong/seke_project.html
defined stage and for the whole bug fixing process. For factors related with source code, we applied Lizard\textsuperscript{d} to parse the source code files in the pre-fix version and compute the value of these factors. For factors related with code changes, first, we developed a tool to parse the detailed commit change log to obtain the index of changed lines of each source code file in the commit. Then we applied the tool Lizard again to parse the source code files in the post-fix version to get their detailed source code structure. Based on the above results, we computed the value of factors related with code changes.

F. Analysis Methods

In order to investigate the relationship between the defined factors and the time of different stages (i.e., $I_{AS}$, $I_{FS}$ and $I_{VS}$) as well as the time of the whole process ($BLT$), two different tests were performed based on the scale of the factors for two subject projects respectively.

For nominal and ordinal factors, we used Kruskal-Wallis test [14] to examine whether there are significant differences in bug fixing time among different values of each factor. Taking PriLevel as an example, we first divided the data items into different groups according to their priority levels (e.g. there are five priority levels in Eclipse), then we extracted the time of corresponding data items for each individual stage as well as the whole process. Kruskal-Wallis test was performed among these groups of data items. P-value < 0.05 denotes that there exit significant differences among the time of data items with different priority levels, which might indicate that the factor of priority level has significant influence on bug fixing time.

For ratio factors, we used Spearman rank correlation test [16] to examine whether there are correlations between the value of certain factors and the bug fixing time. Taking NumCom as an example, we first established two groups of data, one group corresponding to the value of NumCom of each data item, and the other group corresponding to its bug fixing time. Then we performed Spearman rank correlation test for these two groups. P-value < 0.05 denotes NumCom correlates with bug fixing time significantly, further reflecting that it has significant influence on bug fixing time.

III. RESULTS AND ANALYSIS

In this section, we report the results and analysis on the three research questions. As mentioned in Section II.F, the results of Kruskal-Walls test for nominal and ordinal factors and Spearman rank correlation test for ratio factors are presented in TABLE II. We used ‘*’ to denote factors whose correlation is greater than 0.05 and use ‘n/a’ to denote the factors that are not applicable for the test.

A. RQ1: Do the factors have different correlations with the time of different stages?

It is quite common that the factors have different correlations with the time of different stages. From TABLE II, we can observe that all the ratio factors have different correlations with the time of different stages, and many of these factors have large different correlations with the time among stages, e.g., the correlations between NumCom and the time of each stage are 0.873, 0.390 and 0.081 respectively in Linux.

Moreover, we can observe that the value of correlations in $S_A$ is the highest in general, while the value of correlations in $S_V$ is the lowest or even absent. For instance, for both subject projects, factors related with comments (NumCom and LenCom) have higher correlations with $I_{AS}$ (0.541 and 0.531 in Eclipse) and $I_{FS}$ (0.361 and 0.368 in Eclipse) than with $I_{VS}$ (0.204 and * in Eclipse). Another example is the factors related with code changes. They have correlations with $I_{FS}$, but most of their correlations with $I_{VS}$ are quite low or even absent. This is also hold good for the nominal and ordinal factors. For instance, for both subject projects, CompCat and PriLevel have no significant influence on $I_{VS}$, while their influences on $I_{AS}$ are usually significant.

These findings indicate that the influences of factors on bug fixing time are stage related, and during the bug fixing process, the influence of the factors on bug fixing time weaken over time. Moreover, the results verify the necessity of adopting a stage-oriented method for analyzing the relationship between factors and the bug fixing time.

B. RQ2: Do the factors have different correlations between the time of individual stages and the time of the whole process?

It is quite different between the correlations of the factors with the time of stages and those of the whole process. We can observe from TABLE II that, some factors which have no correlations with $BLT$ are actually correlated with certain stages, e.g., source code related factors in both subject projects. Moreover, most of the factors have higher correlations with $I_{AS}$ or $I_{FS}$ than with $BLT$, e.g., CHProd, CHComp and CHAssi in both subject projects. This can be understood that a wrong assignment of location and assignee of bug reports could prolong the time of the assignment stage. However, when putting them to the whole process, their influences become much weaker.

These findings might imply that the influence of the factors on bug fixing time could mainly exist on $S_A$ and $S_V$, which reflects a more accurate relationship between the factors and the time of the bug fixing process. In addition, the results verify the necessity of adopting a stage-oriented analysis again.

C. RQ3: Which factors have the highest correlation with the time of each stage?

In TABLE II, we highlighted the top three correlations for each stage (only one for $S_V$ due to the few and low correlation results), and we can make the following observations.

In $S_A$, the factors with top three highest correlations are NumCom, LenCom and CHAssi in both subject projects. Let’s first focus on NumCom. The high correlation value (0.541 in Eclipse and 0.873 in Linux) means the number of comments could influence the time of the assignment stage. People might expect that more comments signify there is more attention focused on the bug, which should help find the right bug fixer, and thus it results in a shorter assignment time. However, in reality, bug reports with more comments may accompany with

\footnote{http://www.lizard.ws/}
a longer assignment time, especially for Linux. This might because that more comments indicate the bug is difficult to fix, which brings in more people to discuss the resolution in comments. Furthermore, more comments would call for more time and effort from developers to read and make final decisions, which could also potentially extend the bug assignment time. The results in TABLE II also reveal that the number of changes in assignee could influence $I_{AS}$. This is obvious that the change of assignee, commonly known as bug tossing [2, 6], would prolong the bug assignment time. Hence, distributing bug reports to appropriate fixers quickly and precisely can effectively shorten the bug assignment time.

In $S_F$, the factors with top three highest correlations are NumCom, LenCom and AddSLOC_M in both subject projects. Number and length of comments can influence $I_{AS}$. This is obvious that the change of assignee, commonly known as bug tossing [2, 6], would prolong the bug assignment time. Hence, distributing bug reports to appropriate fixers quickly and precisely can effectively shorten the bug assignment time.

In $S_V$, there are almost no correlations between the defined factors and $I_{VS}$. However, we also found that, in the two subject projects, $I_{VS}$ can occupy more than 50% of $BLT$. To investigate the reason for this phenomenon, we randomly sampled 10% bug reports for each project, and manually examined their contents. We found that, almost all the sampled bug reports in both projects changed their status to VERIFIED on the day when the associated version released or a new release tag was assigned. Put another way, the time of verification stage cannot precisely reflect the situation of bug verification activity. That’s why there are few factors that have correlation with the long $I_{VS}$ in both projects.

IV. THREATS TO VALIDITY

In this section, we discuss the threats to validity of our study with respect to construct validity, internal validity and external validity [16].

Construct Validity: The factors used in our study are generally well understood and straightforward to compute based on publicly available datasets of two OSS projects, which enable the replication of this study. Therefore, our study can achieve a strong confidence in construct validity.

Internal Validity: In our study, we relied on the information stored in Bugzilla and Git repositories to construct the link between bug reports and commits. The treatment can obtain precise links at the cost of filtering some bug reports without such a link. This may influence the internal validity. We note that there are techniques to recover the missing link between bug reports and commit (e.g., [20]). In the future, we would like to employ such techniques to help find more links, and further minimize this threat.

External Validity: The subject projects used in our study are highly active in open source community and have been widely used in previous work. Moreover, they are of different domains and use different development languages. However, we have used only two projects, which might make our findings not generalizable enough to other open source projects. This risk could be mitigated by adding more subject projects. This will be explored in our future work.

V. RELATED WORK

A lot of researches have been conducted to empirically investigate the impact factors of bug fixing time. Mockus et al. conducted extensive experiments on several open source projects to identify the factors affecting the time taken to fix a bug. They found that factors such as the number of comments, the length of comments, and the number of assignees significantly affect the time taken to fix a bug.

In a recent study, researchers used data collected from Bugzilla and Git repositories to analyze the impact of various factors on the time taken to fix a bug. They found that factors such as the number of comments, the length of comments, and the number of assignees significantly affect the time taken to fix a bug.
found that in Apache and Mozilla, bugs with higher priority were fixed faster. Panjner [11] found that, in Eclipse project, bugs with little discussion tend to be resolved quickly, however, when bugs receive more conversation, the resolution times become dependent on their severity level. Marks et al. [8] studied bug fixing time in Eclispe and Mozilla, and found that the time taken to report a bug and its location have the most impact on bug fixing time. Anbalagan and Vouk [1] studied the bug reports of Ubuntu project and found that there is a strong linear relationship between the number of users participating in a bug report and the median time taken to fix it. Besides the attribute of bug reports, Saha et al. [12] extracted code change metrics, e.g., number of changed files, for analyzing the reason of long live bugs in four Eclipse projects. Hooimeijer and Weiner [5] measured bug-report-triage time using regression analysis based on bug report metrics. Zhang et al. [17] investigated impact factors in order to understand why delays incurred during bug fixing. These prior researches treated bug fixing as a whole process, or only focused on a particular phase of the process. An ignored phenomenon is that bug fixing is multi-stage process, in which the influence of factors cross the whole fixing process might not remain the same. In contrast to prior researches, our work performed a stage-oriented analysis to explore such situation.

VI. CONCLUSIONS

In this paper, we performed a stage-oriented analysis to investigate the factors impacting bug fixing time based on two large OSS projects. We extracted twenty-four factors from three aspects: bug reports, their associated source code and code changes, and empirically investigated the influence of them on the time of three stages of the bug fixing process and the whole process. Our results show that the influences of factors on bug fixing time are stage related, and thus it is necessary to analyze the relationship between factors and the bug fixing time in a stage-oriented way. We believe our findings can help developers better understand impact factors on bug fixing time, and thus improve bug fixing process management effectively.

In the future, we plan to study more data sources from more projects (both OSS projects and industry oriented projects), and investigate more factors extracted from new dimensions (e.g., participants and code review activities) in order to make more generic findings.

ACKNOWLEDGMENT

This work is sponsored by the NSFC under Grant No. 91218301, 91318302 and 61432001, and this work is also sponsored by the FoShanCAS under Grant No. 2014HT100022.

REFERENCES


