Contents lists available at GrowingScience

International Journal of Data and Network Science

homepage: www.GrowingScience.com/ijds

Understanding and predicting bugs fixed by API-migrations

Nouh Alhindawi[a*], Omar Meqdadi[b], Jamal Alsakran[c], Nader Mohammad Aljawarneh[d] and Hatim S. Migdadi[e]

[a]*Faculty of Sciences and Information Technology, Jadara University, Irbid, Jordan*
[b]*Department of Software Engineering, Jordan University of Science and Technology, Irbid, Jordan*
[c]*Computer Information Science, Higher Colleges of Technology, Fujairah, United Arab Emirates*
[d]*Faculty of Business, Jadara University, Irbid, Jordan*
[e]*The Hashemite University Zarqa, Jordan*

C H R O N I C L E

Article history:
Received: October 8, 2021
Received in revised format: November 28, 2021
Accepted: February 17, 2022
Available online: February 17 2022

Keywords:
Bug fixing
Adaptive change
API migration
Empirical investigation
Bug classification

A B S T R A C T

Bug tracking systems are standard repositories that preserve a large number of uncovered bugs. Once a bug is reported in these repositories, developers search for appropriate changes to fix the bug. However, discovering the changes that can fix the bugs has a negative influence on the schedule and cost of projects. Mainly, fixing bugs could be done by performing some other maintenance changes. In this work, we study and examine the role of adaptive maintenance in the context of API-migration during bug fixing activities through a case study on KOffice, Extragear/graphics, and Open Scene Graph projects. Our goal is to direct developers towards potential bugs early in development which are more likely to be fixed by performing adaptive changes as opposed to other maintenance tasks. We examined the reports of fixed bugs from the bug tracking systems of the studied projects, then we explored several factors related to variant dimensions of the reports and their relevant version history commits, in order to evaluate their effectiveness to decide whether a bug is likely to be fixed by adaptive changes. Our case study results show that bug residency time, textual contents of the report, the component that the bug was found in, and reporter/commenter experience show significant differences between the bugs that are fixed by adaptive changes and other fixed bugs.

1. Introduction

Once someone discovers a bug, he/she creates a bug report in the bug tracking system, such as Bugzilla, describing what the bug is. Then, the bug is assigned to a developer who is responsible for fixing it, and finally, once the bug is resolved, another developer validates the fix and closes the associated report. Several software development activities focus on the process of fixing bugs (Song et al., 2006; Zeng & Rine, 2004). Determining the effort required to fix a bug is an essential stage in project management and planning. Such effort depends on several factors, such as the complexity of both the bug and code (Marks et al., 2011). The primary step of fixing bugs is looking for the appropriate changes that are needed to perform a fix that results in high quality and requires low effort. However, finding the proper changes for such a fix has a negative impact on both the schedule and cost of projects. Fixing a bug could be accomplished by or could be fixed because of performing other perfective or adaptive changes. In the field of perfective maintenance (e.g., adding new features and API-refactoring), there have been prior studies that consider the relationship of such maintenance with the fault and defect fixing activities. Posnet et al. (2013) found that adding new features is positively correlated with defect fixing activities of the Wicket project. The role of API-

* Corresponding author.
E-mail address: hindawnouh@gmail.com (N. Alhindawi)

doi: 10.5267/j.ijdns.2022.2.011

refactoring during the bug fixing activities is investigated in (Kim et al., 2011). The investigation results show that there is an increase in the number of bug fixes after performing an API-level refactoring. Moreover, the results illustrate that there is many revisions that contain both API-refactoring and bug fixing changes. The accommodation of adaptive maintenance for bug fixing activities is not as well-studied. Therefore, we undertook an empirical study of three large open-source projects that previously underwent major adaptive maintenance tasks. We exploited multiple years of both version control systems (CVS) and bug tracking systems (Bugzilla) used by these projects to uncover exactly which bugs got fixed by the undertaken adaptive changes. We then explored several dimensions of the uncovered fixed bugs in order to evaluate factors that affect the decision of whether a bug will be fixed by implementing adaptive changes or not.

This paper makes two contributions. The first contribution is the investigation of the role of API-migration changes on bug-fixing tasks. Thus, we searched for the existence of commits that involve both major API migrations and bug fixes. Our second contribution is building prediction models to help developers flag early on if a bug is more likely to be fixed by performing adaptive changes (in the context of API-migration) rather than other maintenance tasks. Therefore, we studied all bugs that were fixed during adaptive maintenance changes, along with several factors, to help us discriminate bugs fixed by adaptive changes. Below, our study has below research questions:

RQ1: Does API-migration to a new platform facilitate bug fixes?

In this study, we found that the API-migrations improved developer productivity where there were notable bug fixes after performing adaptive maintenance changes.

RQ2: Which factors indicate, with high probability, that a bug will be fixed by adaptive changes?

We found that the bug residency time, textual contents of the bug report, the component that the bug was found in, and reporter/commenter experience are the best indicators of whether a bug will be fixed by adaptive changes or not.

The remainder of this paper is organized as follows. Section 2 describes the approaches we followed in this study. Section 3 shows our case study and the main obtained results. Section 4 mentions several threats to the validity for our study. Section 5 reviews related work followed by Section 6 with the paper conclusions and some directions for future research.

2. Study setup

In this section, we describe our approach. First, we discuss the studied open-source projects, and then we show the approach we follow to identify the bug-fixing revisions of the studied projects. Then, we list the examined factors that can be extracted from the bug reports.

2.1 Studied projects

To conduct our study, we selected three open-source projects that were previously examined in (Meqdadi et al., 2013). Meqdadi et al. (2013) analyzed two KDE packages (KOffice and Extragear/graphics) and OpenSceneGraph (OSG) as study subjects, and manually identified all the undertaken adaptive commits for these systems within a specific period of time. These projects represent prime examples of successful open-source development that involve significant API migration and have changed logs with good quality (Meqdadi et al., 2013). KOffice and Extragrear were undergoing adaptive maintenance between 06/28/2005 and 12/31/2010. The maintenance was due to an update in the QT framework. QT is an open source, platform-independent framework for developing GUIs. QT4 was released in 2004. In 2006, KOffice and Extragear began maintenance for migration from QT3 to QT4. The migration lasted four years, ending in 2010. OpenSceneGraph migrated for similar reasons, but their migration took place between 08/11/2008 and 03/11/2010. Meqdadi et al. (2013) searched through log messages for usages of particular terms (Qt function, interface, and feature names) that had some relevance to what needed to be updated between Qt3 and Qt4. They then read over and inspected the actual commits to make sure they were adaptive change commits. Most of the commits during that time period did not have anything to do with adaptive changes. The other commits addressed corrective maintenance issues or were involved in the adding of new functionality or features to the examined systems. A summary of this is given in Table 1.

Table 1
Adaptive And Non-Adaptive Commits For The Three Systems Over The Given Time Period.

	Koffice	**Extragear/ Graphics**	**OSG**
Adaptive Maintenance Task	Migration from Qt3 to Qt4	Migration from Qt3 to Qt4	Migration to OpenGL 3
Adaptive Task Starting-Date	03/29/2006	11/07/2006	09/18/2008
Adaptive Task Ending-Date	12/31/2010	12/31/2010	12/31/2010
Total Number of Commits	38,980	26,336	4,310
Number of Non-Adaptive Commits	38,849 (99.7%)	26,117 (99.2%)	4,231 (98.2%)
Number of Adaptive Commits	131 (0.3%)	219 (0.8%)	79 (1.8%)

2.2 Identification of Bug Fixing Commits

A number of techniques have been proposed to assist in the recovering of links between fixed bugs and committed changes for open software projects. Traditional heuristics to identify these links depend on the assumption that developers use explicit hints/tags, such as specific keywords (e.g., bug, crash, and fix) and bug ID references, about bug fixing activities in their change commits (Wu et al., 2011). Therefore, recovering the bug-fixing commits from the version history could be accomplished by looking for certain keywords (e.g., fix, bug, and crash) and for bug ID references in change commits (Zimmermann et al., 2007; Pan et al., 2009; Aljawarneh et al., 2021). For instance, shows one of the commits (revision # 1025738) from the Subversion repository of KOffice that was manually labeled as adaptive in (Meqdadi et al., 2013). In this adaptive commit, the developer mentioned the ID of the relevant bug (BUG: 206526) that was fixed using the enhanced version of QPainter class of Qt4 after migrating the system from Qt3 to Qt4. Consequently, to explicitly identify bug-fixing revisions, our methodology follows the approach that was performed in (Wu et al., 2011). This approach is based on searching the commit log messages for valid bug reference numbers.

```
< logentry
   revision="1025738">
<author> berger </author>
<date>2009-09-19T15:14:53.994867Z </date>
<paths>
<path
  kind=""action="M"> /trunk/koffice/krita/ui/canvas/
kis_prescaled_projection.cpp </path>
</paths>
<msg> Fix: the image isn't shown anymore after validating
the preferences and using the QPainter canvas
BUG: 206526
</msg>
</logentry>
</logentry>
```

Fig. 1. A Snippet of an adaptive bug-fixing commit for the KOffice system

3. Study results

We now present the details behind the results and the outcomes of our study.

A. RQ1: Does API-migration to a new platform facilitate bug fixes?

To answer this question, we first used the approach presented in the previous section to identify all bug-fixing commits along with their relevant fixed bugs of the studied projects during the examination period. Afterward, we categorized the uncovered fixing commits into either one of two categories, namely adaptive commits and non-adaptive commits, based on the results of the manual categorization performed in (Meqdadi et al., 2013). The first category contains all commits that fixed bugs by accomplishing adaptive maintenance tasks (e.g., migrating to a new API or compiler). In contrast, the second category contains commits that fixed bug by accomplishing all other maintenance tasks (i.e., adding new features or enhancements tasks). One important note from the results of Meqdadi et al. (2013) is that the adaptive commits normally do not involve large, complex changes to system functionality. Moreover, as an additional verification, we used the *GNU UNIX DIFF* utility to identify changed source code lines occurring in each adaptive commit that was recognized as a bug fix commit. Our verification shows that all the undertaken source code changes of the verified commits were API-migration changes. Given this result, we can form the hypothesis that the first category covers only commits that include both major API migrations and bug fixes. Second, we classified every fixed bug during the examination period into one of two categories: ***Adaptively_Fixed*** and ***Non_Adaptively_Fixed***. The first category contains all bugs that were fixed by performing adaptive changes (e.g., linked to a fix-revision of type adaptive). The second category covers only bugs that were fixed by performing non-adaptive changes (e.g., linked to a fix-revision of type non- adaptive). Table 2 shows the results of using the above approach for the three examined systems, along with number of *Adaptively_Fixed* bugs and *Non_Adaptively_Fixed* bugs. Therefore, the results of the table illustrate the fact that API-migrations through adaptive maintenance fix a set of existing bugs in the open source systems.

We now address a reasonable question concerning the relationship between API-migration and later bug-fixing activities: Are there more bug fixes after committing an API-migration change?

Table 2
Adaptively_Fixed And Non-Adaptively_Fixed Bugs For The Three Systems Over The Given Time Period

	KOffice	Extragear/ Graphics	OSG
Total Number of Fixed Bugs	1234	1107	283
Number of Adaptively_Fixed bugs	57 (4.62%)	86 (7.77%)	34 (12.01%)
Number of Non_Adaptively_Fixed bugs	1177 (95.38%)	1021 (92.23%)	249 (87.99%)

To answer the above question, we need to find all bug-fix revisions (e.g., non-adaptive commits) within N revisions after each adaptive commit where fixing applied to the same file-level locations that were earlier touched by that adaptive commit. To avoid considering the bugs that possibly could be introduced by the undertaken API-migration change of an adaptive commit, where we plan to investigate this type of bug introducing in the future, we only count fixing those bugs that were introduced before the adaptive commit was accomplished. For instance, to find the true benefits of revision *1025738*, see **Fig. 1** on helping later bug fixing, we count the number of bugs that were introduced before *19-09-2009* and fixed within N non-adaptive revisions after committing *1025738*, where the fixing includes changing the file *koffice/ krita/ ui / canvas / kis_prescaled_projection .cpp*.

Based on the results of (Aljawarneh, et al.2021). , we set N to 20. Then, for each adaptive commit, we computed the fix rate (*#fix- revisions / N*). We used these computed fix rates as the data points for classification. To help such classification, we used the 5-point summary approach, the same approach that was used for commit categorization in (Alali et al.,2008). We developed a tool to apply this categorization over the collected adaptive commits and classify them into different categories based on the computed fix rate. Table 3 provides a summary (e.g., the data, quartiles) of the adaptive commit categorization using the fix rate measure for KOffice. The Range column of this table represents the boundaries of each defined region for the corresponding boxplot. The details of the five-point summarization are described elsewhere (Alali et al.,2008). The results show that the API-migrations improve developer productivity where there are notable bug fixes after performing adaptive maintenance changes. For instance, 17.56% of KOffice adaptive commits have fixing rate of 11 % or more.

B. RQ2: Which factors indicate, with high probability, that a bug will be fixed by adaptive changes?

Now, we would like to identify the factor or group of factors that have a significant impact on the recognizing of potential bugs that have the likelihood to be fixed through adaptive changes from the other set of bugs. These factors must represent distinguishing common characteristics and trends that hold for a broad range of *Adaptively_Fixed* bugs and show a statistically significant difference between this type of fixed bugs and the other fixed bugs across several systems. With this information, we can devise our study of which factors to use to detect this type of bug early on.

Table 3
Five-Point Summary Of Adaptive Commits Categorized By Fix Rate For Koffice

		KOffice
Quartile		Fix Rate
Q0 (*Min*)		0
Q1		5%
Q2 (*Median*)		5%
Q3		10%
Q4 (*Max*)		25%
IQR		Q3 - Q1 = 5%
Boxplot	Range	Ratio in adaptive commits
Extra-Small	0% – 5%	60.26 %
Small	6% – 10%	22.18 %
Medium	11% - 17%	12.98 %
Large	18% - 24%	3.82 %
Extra-Large	>= 25%	0.76 %

Prior research is loaded with prediction models regarding the determination and prediction of bugs for various goals. For instance, Shihab.et al (Shihab et al., 2010) identified a set of the top factors that influence the likelihood of a bug being re-opened. In (Valdivia-Garcia et al., 2018), the study results show the major factors that help in predicting blocking bugs which prevent other bugs from being fixed. Sun et.al. (Sun et al., 2011) identified a set of factors to detect duplicate bugs. Kukkar and Mohana (2018) and Lamkanfi et al. (2011) used textual information as a factor to predict bug severities. To come up with a list of factors, we surveyed prior work that had focused on mining bug reports. We leverage the knowledge from numerous prior studies and exposed several factors that could indicate whether a bug will be fixed by adaptive changes or not. Then, we investigated each of these factors and examine if it represents a distinguishing common trend that could recognize *Adaptively_Fixed* bugs from the other bugs. We computed the level of significance of these factors using the Mann-Whitney U non-parametric tests (Conover, 1999). In this statistical testing, a p-value of 0.05 or less represents the significance level. If the p-value is greater than 0.05, then we assume there is no observable difference between *Adaptively_Fixed* and *Non_Adaptively_Fixed* bugs. Based on the results of this test, our investigation classified these factors into one of two categories, namely significant and insignificant factors. The first category includes all factors that could help identify *Adaptively_Fixed* bugs

since they show significant differences between the two types of fixed bugs across all three examined projects. The second category included all other factors that show no significant difference between the two types of fixed bugs.

1- Significant Factors

We now take a closer look at each significant factor and attempt to uncover the significant differences between the *Adaptively_Fixed* and *Non_Adaptively_Fixed* bugs in the terms of each of these factors. The goal is to characterize what a typical *Adaptively_Fixed* bug looks like. Or, more importantly, the goal is to see if a typical *Adaptively_Fixed* bug exists.

- *Bug Residency Time*

First, we examine the interval between the time when a bug was fixed (e.g., resolved and closed), and the time when that bug was initially reported. That is, for each fixed bug linked to a fixing commit in the version history, we examine the elapsed time between its original injection, as given by its associated bug report, and its eventual fix time, as given by its associated fixing commit. This time interval represents the residency time of a bug (Zeng & Rine, 2004). Here, we address the question: do fixed bugs by adaptive commits have a longer residency time than other fixed bugs?

One main advantage of answering the above question is providing a good understanding of the entire life cycle of those bugs that are fixed by the adaptive changes comparing with the other fixed bugs. Table 4 reports the descriptive statistics and the results of the Mann-Whitney test (p-value) obtained when comparing the bug residency time for *Adaptively_Fixed* and *Non_Adaptively_Fixed* bugs. The results demonstrate that the mean time of residency of *Adaptively_Fixed* bugs is approximately 6 months longer than the mean time of residency of the other fixed bugs. Moreover, the results of the Mann-Whitney test always report a statistically significant difference between the residency time of bugs fixed by adaptive changes and other fixed bugs for the three examined systems, where all reported p-values are less than the value of 0.05. Given this trend, we would observe the fact that bugs fixed by adaptive changes often have a longer residency time before being fixed. This fact can be explained by the idea that most of these fixed bugs were originally injected in the system because of existing bugs or missing features in the employed API or compiler release, where developers are unable to wrap around these problems. Consequently, developers wait for a significant improvement of APIs or compilers. For instance, the *QValueList <QString>* does not have a virtual destructor in the release Qt 3.x (QTBUG-8292), and because of that some KOffice code kept crashing and leading to an undefined behavior. This reported KOffice bug has been resolved by using the new class *QList* after releasing the new API Qt 4.8. Once again, a reasonable question to ask is: are our time investigation results analogous and comparable with time interval results of previous empirical studies that investigated residency time of generic fixed bugs? The previous empirical studies indicate that, in most cases, the bug residency time of most bugs that are fixed by nonspecific changes is 3 months (Zeng & Rine, 2004). This finding is comparable with our obtained results regarding the set of bugs that were fixed by non-adaptive changes.

- *Comment Dimension*

Hooimeijer and Weimer (2007) showed that the number of comments associated with a bug report indicates the time it takes to fix it. Additionally, Shihab et al. (2010) used the number of comments as a factor in determining whether a bug will be reopened or not. These prior studies motivate us to investigate if there is any trend regarding the number of comments associated with those bugs that were fixed by adaptive maintenance. As such, for each fixed bug, either by adaptive changes or not, we directly extracted the number of comments associated with it. Table 5 reports the descriptive statistics and the results of the Mann-Whitney test (p-value) obtained when comparing the comment count for *Adaptively_Fixed* and *Non_Adaptively_Fixed* bugs. The results of the Mann-Whitney test report no statistically significant difference between the comment counts of bugs fixed by adaptive changes commits and other fixed bugs, where the p-values are 0.059, 0.054, and 0.047 for the KOffice, Etragear/graphics, and the OSG respectively. Therefore, no solid conclusion can be drawn in this view of the comparison between *Adaptively_Fixed* and *Non_Adaptively_Fixed* bugs in the term of comment count.

Nevertheless, the study results in (Hooimeijer & Weimer, 2007) illustrated that bugs that receive more user attention get fixed faster. Conversely, based on our results of time interval measures of the two types of fixed bugs, which were discussed previously, we would observe that bugs with a large comment count and long residency time are more likely to be fixed by adaptive changes rather than other maintenance tasks. Consequently, the above observation motivates us to investigate the average time interval between successive comments attached to every *Adaptively_Fixed* bug and see if such an average would help in distinguishing this type of fixed bugs from the other fixed bugs. Thus, for each fixed bug, we extracted the time interval between every two successive comments from its report. Then, we computed the corresponding average time interval (AVGct) for the bug. Table 6 reports the descriptive statistics and the results of the Mann-Whitney test (p-value) obtained when comparing the AVGct for *Adaptively_Fixed* and *Non_Adaptively_Fixed* bugs. One important note here, the *Adaptively_Fixed* bugs always show higher values of AVGct. The reason is due to the observation that these bugs received a lower number of comments during a long residency time as we discussed before. Also, the results of the Mann-Whitney test report a statistically significant difference between the AVGct of *Adaptively_Fixed* bugs and other fixed bugs for the three examined systems, where the p-values are always less than the value of 0.05.

Therefore, we would observe that, although there is no significant difference between bugs fixed by adaptive changes and other fixed bugs in the term of comment count, bugs fixed by adaptive changes have a higher time interval between their successive comments.

- *Developer's Experience*

Recently, the research community focused on understanding the relation between the people dimension (e.g., the reporter, commenters, and the fixer) of a given bug report and the bug classification (Al-Bourini et al., 2021). Here, we address the question: is the experience level of developers involved in fixing *Adaptively_Fixed* bugs any different from that of developers involved in other fixing activities? This research question investigates whether a bug fixed through adaptive maintenance requires developers who have higher/lower experience levels with respect to other fixing activities. Since adaptive changes are system-wide maintenance operations that require a developer who has a broad knowledge of the system and APIs, we are hypothesizing that bugs fixed by adaptive changes were performed by developers who have more experience with adaptive maintenance.

Like previous studies, we do not consider the general experience of developers (e.g., total number of days on a project). Instead, we evaluate a developer's experience based on his/her contributions on the undertaken maintenance activities (e.g., adaptive and non-adaptive) and bug fixing accomplishments. We used four different measures to assess developers' contributions. The first measure used to capture the developer's contribution on the undertaken adaptive maintenance is named adaptive experience (AdpExp). This measure is computed for a developer Di as:

$$\text{AdpExp (Di)} = |\text{AdpCommits (Di)}| \,/\, |\text{AdpCommits}|,$$

where |AdpCommits (Di)| is the number of adaptive commits performed by Di and |AdpCommits| is the total number of adaptive commits performed during the examination period. Similarly, we adopted a variant of the first measure, named as adaptive experience (NAdpExp), where the numerator is computed by only taking into account the non-adaptive commits performed by Di, and the denominator represents the total number of non-adaptive commits performed during the examination period. The third measure used to capture the developer's contribution on the undertaken fixing activities that were accomplished through adaptive changes is named as adaptive fixing experience (AFixExp). This measure is computed for a developer Di as:

$$\text{AFixExp (Di)} = |\text{AdpFixBug (Di)}| \,/\, |\text{AdpFixBug}|,$$

where | AdpFixBug (Di)| is the number of bugs that were *Adaptively_Fixed* by Di and | AdpFixBug| is the total number of *Adaptively*_Fixed bugs during the examination period. Also, we adopted a variant of the third measure, named as non-adaptive fixing experience (NAFixExp), where the numerator is computed by only taking into account the bugs fixed by Di, which were performed through non-adaptive changes, and the denominator represents the total number of bugs that were fixed through non-adaptive changes during the examination period.

In essence, given a bug report, our approach works as follows:

1. Extract the bug reporter and the set of all developers who attached a comment to the report. This step is performed by mining the bug tracking system.
2. For every reporter and commenter, compute the four mentioned contribution metrics. This step is performed by mining the author dimension, see **Error! Reference source not found.**, of the commits that are recorded in the version history of the system. Of course, we earlier showed that the commits are classified as either adaptive or non-adaptive, and the set of bug fixing commits (e.g., adaptive and non-adaptive) are also identified for each system.
3. Finally, for each bug report, calculate the following measures:
    a. **S_AdpExp:** Sum of AdpExp of the reporter and all commenters of the bug report.
    b. **S_NAdpExp:** Sum of NAdpExp of the reporter and all commenters of the bug report.
    c. **S_AFixExp:** Sum of AFixExp of the reporter and all commenters of the bug report.
    d. **S_ NAFixExp:** Sum of NAFixExp of the reporter and all commenters of the bug report.

As we can note from the above steps, our computations need data extracted from both the version history and bug tracking system. Since developers could use different login names in different scenarios, we need to map between user accounts in the bug tracking system of a project. For our investigation, we did this mapping manually for each examined system. Table 7 reports the descriptive statistics and the results of the Mann-Whitney test (p-value) obtained when comparing the developers' contributions (measured using four metrics described early) for *Adaptively_Fixed* and *Non_Adaptively_Fixed* bugs. In this table, The four contribution metrics, on all three subject systems, reveal an expected result: developer's experience (e.g., bug reporter, commenter, or fixer) in adaptive maintenance is higher for *Adaptively_Fixed* bugs than for other fixed bugs, while developer's experience in non-adaptive maintenance is lower for Adaptively_Fixed bugs than for other fixed bugs. This difference is also always statistically significant as illustrated by the results of the Mann-Whitney test (p-value < 0.05).

The main explanation of our findings is provided by the results of (Aljawarneh, et al.2020), which illustrate that API-migration changes are complex, critical, and system-wide tasks, and thus performing bug fixing through such migration requires a developer who is an expert with this type of maintenance and understands the new features that exist in the new API version. Therefore, the experience of the reporter/commenter of a bug mirrors better knowledge of the type of changes needed to fix that bug in the future. On the other hand, delving a bit deeper into the people dimension of recorded bug reports, we found that nearly 85% of the fixed bugs through adaptive maintenance hold this observation: the reporter and the fixer of a bug are the same person. Moreover, our investigations show that the developer who fixed a bug by adaptive changes always posted at least one comment in the relevant bug report. Consequently, we would observe that if a bug was reported or commented by adaptive maintenance experts, then it is more likely to be fixed by adaptive changes rather than other maintenance tasks.

Table 4
Bug Residency Time (E.G., #Months) In Adaptively_Fixed And Non_Adaptively_Fixed Bugs: Descriptive Statistics And Mann-Whitney Test (P-Value)

Bug Residency Time							
	Adaptively_Fixed Bugs			Non_Adaptively_Fixed Bugs			p-value
System	Mean	Median	St. Dev.	Mean	Median	St. Dev.	
KOffice	11.23	9.51	6.53	4.51	4.10	4.36	0.026
Extragear/graghics	12.41	9.82	5.13	4.93	3.55	4.71	0.023
OSG	9.83	6.51	3.47	3.16	1.62	3.88	0.032

Table 5
Comment Count In Adaptively_Fixed And Non_Adaptively_Fixed Bugs: Descriptive Statistics And Mann-Whitney Test (P-Value)

Comment Count							
	Adaptively_Fixed Bugs			Non_Adaptively_Fixed Bugs			p-value
System	Mean	Median	St. Dev.	Mean	Median	St. Dev.	
KOffice	3.06	2.00	4.28	3.25	2.00	2.34	0.059
Extragear/graghics	3.41	3.00	2.26	3.72	3.00	2.15	0.054
OSG	3.17	2.00	3.41	3.83	3.00	2.47	0.047

Table 6
AVGCT measure (E.G., #MONTHS) inADAPTIVELY_FIXED and NON_ADAPTIVELY_FIXED BUGS: Decriptive statistics and Mann-Whitney test (P-Value)

AVGct							
	Adaptively_Fixed Bugs			Non_Adaptively_Fixed Bugs			p-value
System	Mean	Median	St. Dev.	Mean	Median	St. Dev.	
KOffice	4.06	2.85	2.09	2.51	1.90	3.22	0.035
Extragear/graghics	4.11	3.73	1.78	1.93	1.32	3.19	0.041
OSG	2.27	1.89	2.62	1.44	1.11	5.56	0.039

- *Location Dimension*

A large body of research focused on considering the component as a factor to predict different aspects of bugs. For instance, Marks et.al (Zeng & Rine, 2004) study the fix-time along three dimensions including the location of the bug (e.g., which component). Sun et al. (2011) included the bug component as a factor in their model to predict duplicate bugs. They found that the component of a bug is one of the most important factors in determining whether the bug will be re-opened or not. Therefore, in this research, we studied the *Adaptively_Fixed* bug along with the location dimension (e.g., component), where we aim at investigating whether this dimension could help in predicting if adaptive changes may fix a given bug or not. The intuition is that the type of fixing changes of a bug would be similar to the common change type of prior bugs in that component. That is, if most bugs in a component were fixed by performing an API-migration task, then a new bug in that component is likely to be fixed by some similar API-migration tasks. Accordingly, in our study, we choose the fix rate (e.g., adaptively and non-adaptively) of each component as a metric to gauge the likelihood of using the adaptive changes to fix a bug in that component. We used two different measures to assess the fix rate of a component. The first measure used to capture the contribution of the adaptive maintenance on fixing prior bugs of a given component is named as adaptive fixing contribution (AdpFixCont) and it is computed for a component Ci as:

$$\text{AdpFixCont (Ci)} = |\text{AdpFixBug (Ci)}| \,/\, |\text{FixBug (Ci)}|,$$

where | AdpFixBug (Ci)| is the number of bugs that were *Adaptively_Fixed* in Ci and | FixBug (Ci)| is the total number of fixed bugs in Ci during the examination period. The second measure used to capture the contribution of the non-adaptive maintenance on fixing prior bugs of a given component is named as non-adaptive fixing contribution (NAdpFixCont). Since, in this study, we classify the fixing changes into either adaptive or non-adaptive, the second measure is computed for a component Ci as:

$$\text{NAdpFixCont (Ci)} = 1 - \text{AdpFixCont (Ci)}.$$

For every component Ci of each studied system, we computed the AdpFixCont (Ci) and NAdpFixCont (Ci). For instance, Fig. 2 shows the computed fix contribution measures over the components of the KOffice system. We can see that most of the components have higher NAdpFixCont than AdpFixCont, while only two components have higher AdpFixCont than NAdpFixCont. That is, we would observe that bugs residing in some specific components are more likely to be fixed through adaptive changes than by the other maintenance changes.

- *Description and Comment Texts*

As we discussed before, the developer who fixed a bug by adaptive changes posted at least one comment related to that fixing activity, which is associated with API-migration tasks. On the other hand, Shihab et al. (2010) identified the bug description and comment text as top factors that influence the likelihood of a bug being re-opened. In (Valdivia-Garcia et al., 2018), the study results show that the bug description and the comment text are important factors in predicting blocking bugs that prevent other bugs from being fixed. These prior studies and our previous results motivate us to address a main question concerning the prediction of bugs that could be fixed by adaptive maintenance changes. Here, we see if there are some terms from the bug description and the comment texts that are strongly related to those bugs being fixed by adaptive changes rather than other bugs. That is, we address the question: do the bug description and the comment text attached to a bug report indicate whether a bug will be fixed by a sort of adaptive changes or not? To answer this question, we need a vocabulary of the most frequent terms used in the bug description and the comment text attached to the reports of *Adaptively_Fixed* bugs over the three examined projects. To deal with textual contents of the description and attached comments of a bug report, a special processing is employed to convert these textual attributes into numerical values. Similar to the prior studies, we used the Naive Bayesian classifier (Meyer & Whateley, 2004) on the description text and attached comments to build a vocabulary of the most frequent terms used in the reports of *Adaptively_Fixed* bugs over the three examined systems. Therefore, we used a training set consisting of 2/3 randomly selected bug reports. Then, two corpora were generated from the training set, where the first corpus (Corpus_A) contains the textual contents of the descriptions and attached comments of the *Adaptively_Fixed* bugs, while the second corpus (Corpus_NA) contains the textual contents of the descriptions and attached comments of the other fixed bugs. Afterward, the Bayesian classifier was trained using the two derived corpora.

The text of the description and attached comments are divided into tokens, where each token represents a single word. The occurrence frequency of each token was computed, and then each word is assigned a probability of being relevant to an *Adaptively_Fixed* bug or *Non_Adaptively_Fixed* bug. The probability P(Wi) is assigned to each word Wi as follow:

- If Wi is only in Corpus_A, then P(Wi) is close to 1.
- If Wi is only in Corpus_NA, then P(Wi) is close to 0.
- Otherwise, if Wi is in both corpora, then its probability is calculated by:

$$P(Wi) = | FreqA(Wi) | / | FreqT(Wi) |.$$

where | FreqA(Wi) | is the occurrence frequency of word (Wi) in Corpus_A and | FreqT(Wi)| is the total occurrence frequency of Wi in the both corpora (occurrence frequency in Corpus_A + occurrence frequency in Corpus_NA). For instance, Table 8 has a list of some words that have a probability close to 1 for the KOffice system ordered by their occurrence frequency Corpus_A(| FreqA(Wi)|), while Table 9 has a list of some words that have a probability close to 0 for the same system ordered by their occurrence frequency in Corpus_NA(e.g., | FreqNA(Wi) |). Once again, we excluded the terms “fix” and “bug” from our consideration as they are expected to be frequent terms in both corpora. As we can observe, some terms are more relevant to *Adaptively_Fixed* bugs than other bugs, where these terms have noteable occurrence frequencies. This gives some credence to the idea of using a set of predefined terms in predicting those bugs being fixed by adaptive changes rather than other changes across a broader set of systems. After the classifiers were trained, for each bug report (Reporti), we combined the description text and the attached comments into one text Ti. Then, we obtained the Bayesian-score of Ti based on the computed probabilities of its words as follow:

$$P(Ti) = \prod P(Wi) / (\prod P(Wi) + \prod (1- P(Wi))).$$

Similar to the prior studies, the calculation of P (Ti) is based on the probabilities of the highest 15 words within the text Ti.
A survey by Bettenburg et al. (Zimmermann et al., 2010) shows that bug reports with code samples are fixed sooner than other bugs. So, we counted the number of semicolons in the bug report (e.g., description and attached comments) of both *Adaptively_Fixed* and *Non_Adaptively_Fixed* bugs, to see any distinguishing trend between the two types of fixed bugs in term of semicolon counting.

2. Significant factor

Table 10 lists the factors that were classified as insignificant by our investigation results, besides their results of the Mann-Whitney test. For instance, as we can note from, the priority measure shows a significant difference between the two types of fixed bugs for only the Extragear/graphics system (e.g., p-value = 0.048), and so we consider this factor as insignificant.

Table 7
Developer's Experience In Adaptively_Fixed And Non_Adaptively_Fixed Bugs: Descriptive Statistics And Mann-Whitney Test (P-Value)

S_AdpExp							
System	Adaptively_Fixed Bugs			Non_Adaptively_Fixed Bugs			p-value
	Mean	Median	St. Dev.	Mean	Median	St. Dev.	
KOffice	0.18	0.15	0.08	0.04	0.01	0.07	0.011
Extragear/graghics	0.13	0.09	0.06	0.03	0.01	0.05	0.012
OSG	0.15	0.12	0.06	0.06	0.02	0.07	0.019
S_NAdpExp							
System	Adaptively_Fixed Bugs			Non_Adaptively_Fixed Bugs			p-value
	Mean	Median	St. Dev.	Mean	Median	St. Dev.	
KOffice	0.04	0.02	0.06	0.09	0.05	0.10	0.013
Extragear/graghics	0.04	0.02	0.05	0.16	0.09	0.19	0.011
OSG	0.03	0.01	0.07	0.12	0.08	0.13	0.018
S_AFixExp							
System	Adaptively_Fixed Bugs			Non_Adaptively_Fixed Bugs			p-value
	Mean	Median	St. Dev.	Mean	Median	St. Dev.	
KOffice	0.20	0.18	0.05	0.03	0.01	0.06	0.022
Extragear/graghics	0.17	0.14	0.09	0.02	0.01	0.06	0.014
OSG	0.19	0.16	0.08	0.04	0.02	0.05	0.017
S_NAFixExp							
System	Adaptively_Fixed Bugs			Non_Adaptively_Fixed Bugs			p-value
	Mean	Median	St. Dev.	Mean	Median	St. Dev.	
KOffice	0.05	0.03	0.08	0.13	0.11	0.17	0.019
Extragear/graghics	0.02	0.01	0.05	0.09	0.06	0.14	0.016
OSG	0.04	0.01	0.07	0.12	0.08	0.16	0.020

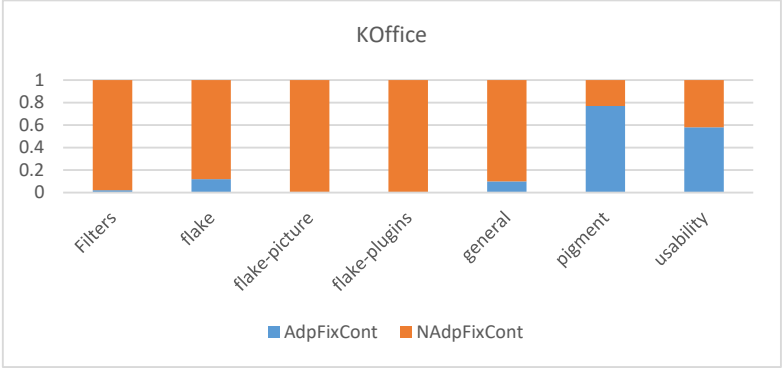

Fig. 2. The distribution of fix rate for bugs over the components of KOffice. The x-axis shows the specific component, while the y-axis shows the computed AdpFixCont and NAdpFixCont

Table 8
Top 15 Words With Probability Close To 1 Ordered By Occurrence Frequency In Corpus_A For The Koffice System

			FreqA(Wi)|		FreqA(Wi)|
Threadpool	17.18%	Platform	10.23%	Port	4.16%
Leak	15.42%	Deprecate	8.81%	Signal	3.88%
Wizard	14.43%	Catch	7.45%	Slot	3.42%
Include	12.89%	Cast	5.33%	Allocate	3.16%
Operator	12.06%	Format	4.47%	Lightweight	3.01%

4. Threats to validity

We now present some possible threats to validity. The systems we studied are all open-source systems. The applicability of our approach for industry/proprietary systems are not claimed in this study, though, and is left for a future investigation. In our study, we used traditional heuristics to identify bugs that were fixed by adaptive changes. The results of conventional heuristics largely depend on the changelog and commit quality. As a result, we could fail to identify all bugs that were fixed by adaptive changes. Moreover, developers have non-standard patterns as when to commit. Some commit every small successful change early due to time constraints, where others wait until the completion of the whole problem.

5. Conclusion

The paper presents a new model based on the decision trees for predicting whether or not a bug will be fixed by implementing adaptive changes in the context of API-migration. The model locates bugs of interest in the bug-tracking systems. A set of

factors extracted from the version control and bug repositories of three large open-source systems have been evaluated. Our results show that a set of factors show significant differences between the bugs that are fixed by adaptive changes and other fixed bugs. We intend to investigate this path in future research. We also plan to expand our study by investigating more projects and figuring out if our proposed model is valid for industrial projects. Moreover, we plan to construct a golden set of API-transformational rules to automatically fix some system bugs.

Table 9
Top 15 Words With Probability Close To 0 Ordered By Occurrence Frequency In Corpus_Na For The Koffice System.

	FreqNA(Wi)|		| FreqNA(Wi)|		| FreqNA(Wi)|
Work	14.75%	Pattern	7.45%	Build	3.52%
Break	11.61%	Overrun	7.31%	Load	3.09 %
Debug	10.17%	Disable	7.05%	Patch	2.76%
Duplicate	8.02%	Message	6.30%	Query	2.42%
Typo	7.78%	behavior	5.81%	Background	1.83%

Table 10
The summary of factors that show no significant difference between adaptively_fixed and non_adaptively_fixed bugs

Name	**Factor Description**	**Computed p-value**		
		KOffice	**Extragear /Graphics**	**OSG**
Severity	The severity of a reported bug that describes the impact of it. The severity of a bug is represented by an integer number with a range from 1 (e.g., low) to 7 (e.g., high)	0.054	0.071	0.066
Priority	The priority of a reported bug that shows the order in which a bug should be focused on with respect to other bugs. The priority of a bug is represented by an integer number with a range from 1 (e.g., high) to 5 (e.g., low).	0.067	0.048	0.059
Priority Changed	Shows whether or not the priority of a bug has been changed after the initial report. The value of 1 means the priority has been changed, and 0 otherwise	0.054	0.071	0.061
Number in the CC list	The number of developers in the CC list of a bug report	0.062	0.055	0.041
Description size	The total number of words in the description of a bug in its report	0.071	0.064	0.053
Comment size	The total number of words of all comments attached to a bug report	0.069	0.044	0.058
Comment Count	The total number of comments attached to a bug report	0.058	0.054	0.049
Reporting Time	The time when the bug was reported. The time is measured as: 1(morning), 2 (day), and 3(night).	0.052	0.055	0.052
Reporting Date	The date when the bug was reported. We use two metrics to capture the reporting date: month and year	0.061	0.055	0.055

References

Alali, A., Kagdi, H., & Maletic, J. I. (2008, June). What's a typical commit? a characterization of open source software repositories. *In 2008 16th IEEE international conference on program comprehension* (pp. 182-191). IEEE.

Al-Bourini, F. A., Aljawarneh, N. M., Almaaitah, M. F., Altahat, S., Alomari, Z. S., & Sokiyna, M. Y. (2021). The Role of E-Word of Mouth in the Relationship between Online Destination Image, E-satisfaction, E-Trust & E-Service Quality for International Tourists Perception. *Journal of Information Technology Management*, *13*(Special Issue: Big Data Analytics and Management in Internet of Things), 92-110.

Aljawarneh, N. M., Abd kader Alomari, K., Alomari, Z. S., & Taha, O. (2020). Cyber incivility and knowledge hoarding: Does interactional justice matter?. *VINE Journal of Information and Knowledge Management Systems.*

Aljawarneh, N., Mahafzah, A., Altahaa, S., Alzboun, E., & Harafsheh, I. (2021). Online sales system and organization outcome. *International Journal of Data and Network Science*, *5*(3), 485-494.

Aljawarneh, N., Taamneh, M., Alhndawi, N., Alomari, K., & Masad, F. (2021). Fog computing-based logistic supply chain management and organizational agility: The mediating role of user satisfaction. *Uncertain Supply Chain Management*, *9*(3), 767-778.

Alsafadi, Y., Aljawarneh, N., Çağlar, D., Bayram, P., & Zoubi, K. (2020). The mediating impact of entrepreneurs among administrative entrepreneurship, imitative entrepreneurship and acquisitive entrepreneurship on creativity. *Management Science Letters, 10*(15), 3571-3576.

Conover, W. J. (1999). Practical nonparametric statistics (Vol. 350). john wiley & sons.

Hooimeijer, P., & Weimer, W. (2007, November). Modeling bug report quality. *In Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering* (pp. 34-43).

Kim, M., Cai, D., & Kim, S. (2011, May). An empirical investigation into the role of API-level refactorings during software evolution. *In Proceedings of the 33rd International Conference on Software Engineering* (pp. 151-160).

Kukkar, A., & Mohana, R. (2018). A supervised bug report classification with incorporate and textual field knowledge. *Procedia Computer Science, 132*, 352-361.

Lamkanfi, A., Demeyer, S., Soetens, Q. D., & Verdonck, T. (2011, March). Comparing mining algorithms for predicting the severity of a reported bug. *In 2011 15th European Conference on Software Maintenance and Reengineering* (pp. 249-258). IEEE.

Marks, L., Zou, Y., & Hassan, A. E. (2011, September). Studying the fix-time for bugs in large open source projects. *In Proceedings of the 7th International Conference on Predictive Models in Software Engineering* (pp. 1-8).

Menzies, T., & Marcus, A. (2008, September). Automated severity assessment of software defect reports. *In 2008 IEEE International Conference on Software Maintenance* (pp. 346-355). IEEE.

Meqdadi, O., Alhindawi, N., Collard, M. L., & Maletic, J. I. (2013, September). Towards understanding large-scale adaptive changes from version histories. *In 2013 IEEE International Conference on Software Maintenance* (pp. 416-419). IEEE.

Meyer, T. A., & Whateley, B. (2004, July). SpamBayes: Effective open-source, Bayesian based, email classification system. In CEAS.

Pan, K., Kim, S., & Whitehead, E. J. (2009). Toward an understanding of bug fix patterns. *Empirical Software Engineering*, 14(3), 286-315.

Posnett, D., Hindle, A., & Devanbu, P. (2011, October). Got issues? do new features and code improvements affect defects?. *In 2011 18th Working Conference on Reverse Engineering* (pp. 211-215). IEEE.

Shihab, E., Ihara, A., Kamei, Y., Ibrahim, W. M., Ohira, M., Adams, B., ... & Matsumoto, K. I. (2010, October). Predicting re-opened bugs: A case study on the eclipse project. *In 2010 17th Working Conference on Reverse Engineering* (pp. 249-258). IEEE.

Song, Q., Shepperd, M., Cartwright, M., & Mair, C. (2006). Software defect association mining and defect correction effort prediction. *IEEE Transactions on Software Engineering*, 32(2), 69-82.

Sun, C., Lo, D., Khoo, S. C., & Jiang, J. (2011, November). Towards more accurate retrieval of duplicate bug reports. *In 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)* (pp. 253-262). IEEE.

Valdivia-Garcia, H., Shihab, E., & Nagappan, M. (2018). Characterizing and predicting blocking bugs in open source projects. *Journal of Systems and Software*, 143, 44-58.

Wu, R., Zhang, H., Kim, S., & Cheung, S. C. (2011, September). Relink: recovering links between bugs and changes. *In Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering* (pp. 15-25).

Zeng, H., & Rine, D. (2004, September). Estimation of software defects fix effort using neural networks. *In Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004. COMPSAC 2004*. (Vol. 2, pp. 20-21). IEEE.

Zimmermann, T., Premraj, R., & Zeller, A. (2007, May). Predicting defects for eclipse. *In Third International Workshop on Predictor Models in Software Engineering (PROMISE'07: ICSE Workshops 2007)* (pp. 9-9). IEEE.