Reopened bug prediction

Talal EL Afchal
Università della Svizzera italiana (USI), Switzerland
talal.el.afchal@usi.ch

ABSTRACT

Reopened bugs are an undesirable problem, since they have a negative impact on the software development life cycle. In fact fixing a reopened bug generally requires double effort and time.
In order to predict whether a bug can be reopened after it has been fixed, we collect several information related to the involved programmers and to the code quality and complexity, and we investigate their impact on the bug reopening prediction.
In this project we analyze three GitHub public repositories *Elasctic search*, *Spring boot*, and *RxJava*. We extracted all commits and bug issues, and then we collect and create the features in which we are interested. These features can be grouped in three categories 1) developers experience 2) bug report 3) code quality.
The results of our study point out that we can predict a reopen bug with 77.2% *presison* and 81.8% *recall.*

1 INTRODUCTION

In general when a bug is detected, a *bug issue* with some description is created. The developers can discuss how to solve the bug by posting their comments to the bug issue. Once the bug is solved a commit with the fix is pushed and the bug issue is closed. At this point the programmers assume that the fix has solved the bug, and they can continue implementing other tasks on the top of the fixed bug. But what are the consequences if the bug is not fixed as the programmers assumed? A reopen bug requires double effort, the programmers need to take a step back and try to understand again what the code is doing in order to locate the bug and fix it. And more than that the tasks which has been implemented can be affected by the bug and a new revision can be required. A bug reopen predictor tool which can calculate the likelihood that the bug issue can be reopened can save time and effort by suggesting the developers to add more test cases. The bug issue and the fixing commit can tell us a lot about the bug. Our tool extracts several features as mentioned before they can be classified in 3 categories.

- **Developers experience**: We believe that several aspects of the developers involved in a bug issue have a big impact on the prediction. From these aspects we are interested in the developer experience in opening a bug or fixing one, and more than this we investigate if the developers have already collaborate in the past.
- **Bug report**: A bug report description which contains the steps to reproduce the bug, and describes the expected behavior, combined with the number of developers comments influence the bug issue understanding.
- **Code quality**: In many bug reopen prediction research studies, the code complexity and quality have been ignored, but in our tool we also explore the code quality and we focus on the complexity changes before and after a fix commit.

Table 1: Number of *Reopened* and ¬*Reopened* bugs

	¬*Reopened*	*Reopened*
ElasticSearch	4019	105
Linked	**1795**	**42**
Rx	531	12
Linked	**259**	**7**
Spring	882	112
Linked	**428**	**42**

In this study we selected three projects written in Java from the GitHub repositories. *Spring boot*, and *RxJava*. As we can see in Table 1, in the ElasticSearch repository we found 105 reopened bug and 4019 which are not reopened, and we where able to link only 1795 not reopened and 42 reopened to their fix commit. It is evident that in all the three repositories we have a balancing problem, since the number of not reopened bug is much higher than the opened on. We face this problem with 3 sampling techniques, and we achieved 77.2% of precision and 81.8% recall in predicting that a bug will be reopened.

2 RELATED WORK

There are two papers closely related to our research. Zimmermann et al [1] analyze and categorize reopened bugs in the Microsoft Windows operating systems Windows Vista and 7, and he collected 394 developers feedback in order to understand the important factors that causes bug reopens. Zimmermann et al. main focus was to investigate the different reasons for bug reopening.
On the other hand Shihab et al.[2] analyzed work habits, bug report, to figure out which factors influence a bug reopening and a reopened bug prediction model using decision trees was built. Shihab et al. found that the comment and description text, time to resolve the bug were the leading factors that caused bug reopening. Our studies differs from Zimmermann et al. since we study an open source data and our main focus is to predict a bug reopening, where Zimmermann et al. main focus stopped at understanding the factors that cause a reopening bug.
And our study complements Shihab et al.[2] since we add more features as the code complexity and the code quality in order to predict a bug reopening, and these features are not taken in consideration in Shihab et al. work. And our study differs in how we process the comment and description features, where we try to understand if the steps to reproduce the bug and the expected behavior are present instead of applying Shihab et al. correlation coefficient algorithm.

3 STUDY DESIGN

Our research question is measuring the performance of bug reopening predictor:

- **RQ**: *To what extent is it possible to predict reopened bugs?*

3.1 Context Selection

In order to answer our research question, we analyzed three public Github repositories: *Elasctic search*, *Spring boot*, and *RxJava*. We decide to analyze these repositories since they have many bug issues which increase the probability of finding reopened bug issues and this is an important factor in our study.
In these repositories both issues and commits are available on the Github platform, which is an important fact, since we analyze both of them and we need to link the bug issue to the fix commit and having them on the same platform is an advantage.
These repositories are Java repository and we decide to analyze only projects written in Java since we calculate the CK metric by using the *mauricioaniche* library, which calculates the CK metric only for Java files.

3.2 Data Extraction Process

We apply the same procedure on each repository to extract data. The first step is to get all the issues (closed and opened) and we filter the one with a *bug* label. For each bug issue we extract the features we are interested in. The list of features is shown in Table2. In the next step we check if the developer mentioned how to reproduce the bug, and it is done by looking if one of the words {**step**, **can**}, and one the words {**reproduce**, **recreate**, **observed**} are present in the bug description. We also check if the programmer mentioned the expected behavior and it is done by looking if one of the words {**should**, **must**}, and one the words {**behavior**, **instead of**} are present in the bug description. The last sept is to count the number of words present in the description and the number of developers involved in the discussion comments.
We clone the repository and as a first step we extract the log file which contains the *commit sha*, *commit message*, *author*, *commit date*, and the *modified files*. In the next step we map each fixing commit to the bug issue. To map a fix commit to a bug issue it has to satisfy all the following rules :

- The commit date must be earlier than the bug issue closing date.
- The commit date must be after the bug issue opening date.
- The commit message has to contain the bug issue number.

These rules were very strict and we were able to map a very small number of reopened bugs, therefor we decide to make our mapping rules more flexible and we modify the rule *The commit message has to contain the bug issue number* to *The commit message has to contain the bug issue number || the developer who closed the bug issue has pushed the commit.* Now that we have extracted all the information from the commits and bug issues, we can extract the social features which are shown in Table 3.
As a last step we calculate the *mean* and *median* of the *CK quality metric*, the *readability*, and the *comments density* of the modified files before the fix commit, and we calculate the *mean* and *median* of the*LOC* after the fix commit and the Δ *complexity* before and after the fix commit.

Table 2: List of extracted features from issues

Feature	Type	Description
IssueId	String	The issue id
Number	String	The issue number
Title	String	The issue title
Description	String	The issue description
CommentAuthor	String	The comment author
CommentDate	String	The comment date
CommentText	String	The comment text
OpenedBy	Numeric	The developer who opened the issue
OpenedOn	Numeric	The date of issue opening
Reopen	Boolean	Check is the issue is reopened
ReopenOn	Numeric	The date of reopening bug
ClosedOn	Numeric	The date of closing the issue
ClosedBy	String	The developer who closed the issue

Table 3: List of social features

Feature	Type	Description
OpenerCommitExperience	Numerical	Overall experience of the developer opening the bug
OpenerFixingExperience	Numerical	Bug fixing experience of the developer opening the bug
FixerCommitExperience	Numerical	Overall experience of the developer fixing the bug
FixerFixingExperience	Numerical	Bug fixing experience of the developer fixing the bug
SocialStrength	Numerical	Number of pairs of developers who took part in the issue discussion that already collaborated in the past

3.3 Analysis Method

The collected data in Table 4 is passed to a data mining software (*Weka*) which apply the *RandomForest* machine learning algorithm, with 10 *folds Cross-validation* technique. *Weka* outputs the *accuracy*, *precision* and *recall* of our prediction. But since the number of reopened bugs is too small respect to the not reopened one, as has we can see in Table1 is causing an imbalance problem. Our approach is to explore three different sampling techniques.

(1) We select a set which contains all reopened bugs and we select randomly a set with same size which contains the not reopened bugs. We repeat this step 10 times which generates 10 different not reopened bugs sets, we pass the reopened bugs set and each of the not reopened bugs sets to *Weka* which gives us 10 predictions. As a last step we calculate the *average, median* and the *standard deviation.* The result is shown in Table 5
(2) Instead of selecting 2 sets with the same size, the not reopened bugs set will be twice bigger than the reopened bug set. As we did previously, we create 10 sets and we

calculate the *average, median* and the *standard deviation*. The result is shown in Table 6

(3) We do the same steps that we did in point (2) with one difference, we do an oversampling where we duplicate the set of reopened bugs by creating artificial samples with similar features to the original ones. The result is shown in Table 7

Table 4: All features

Feature	Type	Description
StepToReproduce	Boolean	Step to reproduce is mentioned
ExpectedBehavior	Boolean	Expected behavior is mentioned
DescriptionLength	Numerical	The description words count
Commenters#	Numerical	Number of programmers who has post a comment
OpenerCommitExp	Numerical	Overall experience of the developer opening the bug
OpenerFixingExp	Numerical	Bug fixing experience of the developer opening the bug
FixerCommitExp	Numerical	Overall experience of the developer fixing the bug
FixerFixingExp	Numerical	Bug fixing experience of the developer fixing the bug
SocialStrength	Numerical	Number of pairs of developers who took part in the issue discussion that already collaborated in the past
LOCPreFix	Numerical	Line of code before the fix
Δcomplexity	Numerical	Δ Cyclomatic complexity
Readability	Numerical	Code readability
CD	Numerical	Code density
CBO	Numerical	Coupling between objects
DIT	Numerical	Depth Inheritance Tree
NOC	Numerical	Counts the number of children a class has
NOF	Numerical	Counts the number of fields in a class
NOPF	Numerical	Counts only the public fields
NOSF	Numerical	Counts only the static fields
NOM	Numerical	Counts the number of methods
NOPM	Numerical	Counts only the public methods
NOSM	Numerical	Counts only the static methods
NOSI	Numerical	Counts the number of invocations to static methods
RFC	Numerical	Counts the number of unique method invocations in a class
WMC	Numerical	It counts the number of branch instructions in a class
LOC	Numerical	It counts the lines of count, ignoring empty lines
LCOM	Numerical	Lack of Cohesion of Methods
reopened	Boolean	The is reopened or not

Table 5: Undersampling 100-100

	Average	Median	Std Dev
Accuracy	0.665	0.645	0.041
Precision-$\bar{R}$	0.664	0.655	0.038
Recall-$\bar{R}$	0.667	0.659	0.052
F-Measure-$\bar{R}$	0.666	0.648	0.043
Precision-R	0.667	0.648	0.044
Recall-R	0.662	0.67	0.042
F-Measure-R	0.664	0.656	0.04

Table 6: Undersampling 200-100

	Average	Median	Std Dev
Accuracy	0.676	0.673	0.023
Precision-$\bar{R}$	0.718	0.718	0.014
Recall-$\bar{R}$	0.849	0.846	0.024
F-Measure-$\bar{R}$	0.778	0.776	0.016
Precision-R	0.525	0.517	0.058
Recall-R	0.333	0.335	0.040
F-Measure-R	0.407	0.408	0.043

Table 7: Oversampling 100-100

	Average	Median	Std Dev
Accuracy	0.788	0.784	0.019
Precision-$\bar{R}$	0.807	0.806	0.025
Recall-$\bar{R}$	0.757	0.758	0.031
F-Measure-$\bar{R}$	0.781	0.778	0.020
Precision-R	0.772	0.766	0.022
Recall-R	0.818	0.821	0.030
F-Measure-R	0.794	0.793	0.019

3.4 Replication Package

A link to a zip file containing all the data and *R* scripts used to run the study. The zip file must contain a README.txt file explaining the data and the scripts.

4 RESULTS DISCUSSION

Discussion of the achieved results. Tip: define a subsection for each research question; conclude the results discussion for each research question by explicitly answering it.

5 THREATS TO VALIDITY

Discuss the threats that could affect the validity of the reported results.

6 CONCLUSION AND FUTURE WORK

Summarize your findings, highlight ideas for future work (No more than one column).

REFERENCES

[1] P. Guo T. Zimmermann, N. Nagappan and B. Murphy. 2012. Characterizing and predicting which bugs get reopened. *Proceedings of the 34th International*

Conference on Software Engineering 34 (2012).

[2] E.Shihab Xinyu Wang B.Zhou X.Xia, D.Lo. 2014. *Automatic, high accuracy prediction of reopened bugs.* Springer Science+Business Media. 249–258 pages.