e-Informatica Software Engineering Journal, Volume 11, Issue 1, 2017, pages: 103–116, DOI 10.5277/e-Inf170105

NRFixer: Sentiment Based Model for Predicting the Fixability of Non-Reproducible Bugs

Anjali Goyal[*], Neetu Sardana[*]
[*]*Jaypee Institute of Information Technology, Noida, India*
anjaligoyal19@yahoo.in, neetu.sardana@jiit.ac.in

Abstract

Software maintenance is an essential step in software development life cycle. Nowadays, software companies spend approximately 45% of total cost in maintenance activities. Large software projects maintain bug repositories to collect, organize and resolve bug reports. Sometimes it is difficult to reproduce the reported bug with the information present in a bug report and thus this bug is marked with resolution non-reproducible (NR). When NR bugs are reconsidered, a few of them might get fixed (NR-to-fix) leaving the others with the same resolution (NR). To analyse the behaviour of developers towards NR-to-fix and NR bugs, the sentiment analysis of NR bug report textual contents has been conducted. The sentiment analysis of bug reports shows that NR bugs' sentiments incline towards more negativity than reproducible bugs. Also, there is a noticeable opinion drift found in the sentiments of NR-to-fix bug reports. Observations driven from this analysis were an inspiration to develop a model that can judge the fixability of NR bugs. Thus a framework, NRFixer, which predicts the probability of NR bug fixation, is proposed. NRFixer was evaluated with two dimensions. The first dimension considers meta-fields of bug reports (model-1) and the other dimension additionally incorporates the sentiments (model-2) of developers for prediction. Both models were compared using various machine learning classifiers (Zero-R, Naïve Bayes, J48, random tree and random forest). The bug reports of Firefox and Eclipse projects were used to test NRFixer. In Firefox and Eclipse projects, J48 and Naïve Bayes classifiers achieve the best prediction accuracy, respectively. It was observed that the inclusion of sentiments in the prediction model shows a rise in the prediction accuracy ranging from 2 to 5% for various classifiers.

Keywords: bug report, bug triaging, non-reproducible bugs, sentiment analysis, mining software repositories

1. Introduction

A software bug is an error or fault in a program which causes the software to behave in unintended ways. Software bugs are usually annoying and inconvenient for developers, often leading to serious consequences. Large software projects use bug tracking repositories where the users and developers report all the bugs they encounter. The developers try to reproduce the bugs with the help of information provided by a reporter in a bug report and then make the required corrections in the source code to rectify the issue. However, sometimes it is not possible to reproduce the reported bug with the information specified in a bug report. In such a scenario, the bug is marked with resolution "Non-Reproducible" or "works for me".

NR bugs account for approximately 17% of all bug reports and 3% of these bugs are later marked as fixed [1]. There could be various reasons behind this fixation of NR bugs. It may be due to any new code patch that might be made available by the reporter, user or developer which could help to reproduce the cause of a bug, or there may be various ways of fixing it. Thus the choice of the solution tested by the developer to reproduce or fix the bug could be wrong [2] and either a new solution or a new developer can reproduce and fix the NR bug.

Another reason could be that the developer had initially marked the bug as NR erroneously due to negligence or, possibly, in reluctance to reduce his or her workload. Thus at later times, a new or previously assigned developer solves the bug. If there was a mechanism which could provide information to the developer beforehand whether the bug report currently marked as NR would be fixed in the future or not, it would not only provide insights to a triager, but also help developers to predict if a bug report marked as NR could be fixed in the future or not. This would save time, effort and cost incurred in those NR bugs in the case of which there is low probability of fixing. With the use of such a mechanism, developers and a triager can actually devote their precious time and efforts to those bugs that are regarded as fixable by the proposed mechanism. This would also raise the level of interest among developers towards NR bugs.

The objective of this work is to establish if it is possible that a bug report, currently marked as NR, will get fixed in the future or not. Thus, the investigation of bug reports is carried out at two different levels. At the first level, the sentiments of comment messages in NR bugs are mined to investigate whether there is any difference between the sentiments of NR-to-fix bug reports and NR bug reports that do not get fixed. At the second level, the NRFixer framework that predicts the probability of NR bug fixation is proposed. NRFixer is evaluated with two dimensions. The first dimension considers the meta-fields of bug reports, such as a component, hardware, a platform, etc., to develop a prediction model-1. Another dimension additionally incorporates sentiments along with the existing meta-fields of a bug report to develop prediction model-2. In this work, the investigations were carried out with reference to six research questions (RQs) to attain two research objectives (ROs). RO1 investigates the sentiment analysis of bug reports using (RQ1–RQ4), whereas RO2 examines the performance of NRFixer using (RQ5–RQ6).

Research Objective 1 (RO1): Exploring sentiments in bug reports.

- RQ1. Do sentiments exist in the NR bug reports?
- RQ2. What is the difference between sentiments of reproducible (R) bugs and NR bugs?
- RQ3. Do the sentiments of developers vary in different categories of NR bugs?
- RQ4. Compare the developer's sentiments for the bug report passing through the stages: 'Newbug-to-NR' and 'NR-to-fix'?

Research Objective 2 (RO2): The fixability prediction of NR bugs.

- RQ5. What is the probability of NR fixing with the use of different classifiers?
- RQ6. Does the inclusion of the category of an NR bug and the sentiments of developers affect the accuracy of a prediction model?

For experimental evaluation, bug reports extracted from Eclipse and Firefox projects of Bugzilla repository were used to gauge the presence of sentiments. Bugzilla is the most popular open source bug repository used by different popular projects, such as Firefox, Eclipse, Linux, etc. Both prediction models (model-1 and model-2) were evaluated using various machine learning classifiers. It was observed that model-1 achieved an accuracy of 70.2% for Firefox and 66.4% for the Eclipse project. The inclusion of sentiments (model-2) further achieved an increase of 2–5% in precision values.

The remainder of this paper is structured as follows. Section 2 illustrates the related work. Section 3 discusses certain preliminaries. Section 4 presents the proposed architecture for NRFixer. Section 5 provides experimental details. Section 6 presents the results of experimental evaluation. Section 7 discusses various threats to validity. Finally, section 8 concludes the paper and discusses future directions for research.

2. Background

This section presents the previous works closely related to the areas: a) Sentiment analysis of bug reports and b) Prediction models in bug repositories.

2.1. Sentiment analysis of bug reports

Sentiment analysis is becoming an important area in the field of natural language analysis.

It comprehends the natural language processing, text analysis, computational logistics with the human psychology to gain an individual's attitude to or feeling about a particular situation or product. It is the "task of identifying positive and negative opinions, emotions and evaluations" [3]. Jurado et al. [4] confirmed that developers do leave sentiments in the textual units of issue repositories. Murgia et al. [5] analysed the existence of emotions in software artefacts, such as issue reports. Their finding confirms the presence of various emotions, such as joy, love, surprise, anger, sadness and fear in issue reports. They also reported that emotions have an impact on software development activities, such as bug fixing. A developer possessing negative emotions may not be able to fix the bug and, thus, it should be assigned to some other developer. Tourani et al. [6] evaluated automatic sentiment analysis in open source mailing lists of the Apache project. The manual study of the emails performed by them contains 19.77% positive sentiments and 11.27% contains negative sentiments. Garcia et al. [7] presented a case study on the Gentoo project of the Bugzilla repository to mine the role of emotions in the contributor's activities. Their study found that a contributor become inactive after experiencing strong positive or negative emotions.

Pletea et al. [8] gauged the presence of emotions in the security related discussions on the GitHub repository. They found that more negative emotions are expressed in security related discussions than in other discussions. The results obtained reflect the reluctance of developers towards the sensitive issue of security. Guzman et al. [9] analysed the sentiments of commit comments in the GitHub repository with respect to four parameters: programming language, day of the week and time of writing the comment, geographic distribution of a team and project approval. Destefanis et al. [10] showed that politeness in developers' comments affects the time to fix an issue.

2.2. Prediction models in bug repositories

As for the prediction model, Garcia et al. [11] built a model to predict blocking bugs. This work is similar to the work on using various machine learning classifiers for prediction. They utilized 14 different parameters to discriminate between blocking and non-blocking bugs and then compared the efficiency of a decision tree, Naïve Bayes, kNN, random forest and Zero-R classifier. They achieved an F-measure of 15–42% by tenfold cross validation on various different bug datasets. The prediction analysis described in this paper is similar to their work. However, there is a difference in the manner of predicting the NR bugs that may get fixed in the future.

Shihab et al. [12] addressed the nature of bugs that get reopened. They used 22 different factors categorized under four dimensions: (1) the work habits dimension, (2) the bug report dimension, (3) the bug fix dimension, and (4) the team dimension. Their model achieved a precision of 52.1% to 78.6% and a recall of 70.5% to 94.1% when predicting whether a bug will be reopened or not. They also found a comment text and the last status to be the most influential factors for predicting the possibility of reopening. Hewett et al. [13] predicted the time required to repair software defects. Their model achieved an accuracy of 93.44% on medical software system dataset. Guo et al. [14] proposed a statistical model to predict the possibility of fixing a newly arrived bug. Their model achieved 68% precision and 64% recall on the Windows Vista project. They further validated their model by conducting a survey among 1773 Microsoft employees. Zimmermann et al. [15] also investigated and characterized the reopened bugs in Microsoft Windows to find the possible causes of reopening bugs and their impact.

3. Preliminaries

This section summarizes the basic information about a bug report, sentiment analysis technique and various machine learning classifiers used in this paper.

3.1. Bug report

A bug report is a document containing complete specification related to a bug. A bug report may be created by an end user, developer or beta tester of the software project. A bug report constitutes various predefined meta-fields and free form textual contents. The predefined meta-fields include bug id, product, component, operating system, platform, milestone, severity, version, status, resolution, reporter, reported date and time, assigned to, etc. The textual contents include keywords, summary (or tagline), description and comments. “Summary” refers to the one-line short definition about the bug. “Description” refers to the complete detailed specification submitted by the reporter regarding the submitted bug. It forms the main body of the bug report that generally incorporates the steps to reproduce the issue. “Comments” refers to the open discussion by a group of people to discuss and review the solutions for the reported bug. This group of people generally comprehends some expertise in the related area of the bug.

3.2. Sentiment analysis

Sentiment analysis is a technique to extract, identify or characterize the sentimental content of a text unit. It assigns a quantitative value representing the contextual polarity of the text. To analyse the sentiments in bug reports natural language text processing (NLTK) toolkit was used [16]. NLTK takes a text unit as an input and performs a two-level classification. Level 1 determines whether the text is neutral or polar. A text unit may or may not contain sentiments. If the probability of the lack of sentiment is greater than 0.5, the text is labelled as neutral. Otherwise, if the probability of the presence of sentiments is greater than 0.5, the text is labelled as polar, and the second level classification is performed to determine whether the text expresses positive or negative sentiment. The label with higher probability is finally assigned to the input text.

3.3. Machine learning classifiers

1. **Zero-R**: Zero-R (or no rule classifier) is the simplest classification algorithm. It always predicts the majority class present in the training dataset. Although it has no predictability power, it is useful in the determination of the baseline performance as a benchmark. In this study, during each fold of cross validation, the Zero-R classifier predicts the majority class among NR-to-NR and NR-to-fix classes during that fold. The individual fold efficiencies related to NR-to-fix class are aggregated to compute the overall efficiency of the Zero-R classifier for NR-to-fix class.
2. **Naïve Bayes**: Naïve Bayes [17] is a simple probabilistic classification algorithm based on Bayes' theorem. It classifies a new record $x = < x_1, \ldots, x_p >$ to class k that maximizes the conditional probability:

$$P(C = k/X) = < x_1, \ldots, x_p >$$

Under the assumption that the factors are randomly independent of each other, the Naïve Bayes classifier can be re-written as: Here, $P(C = k)$ is known as the class prior probability and can be approximated with the percentage of training files marked with label k. The likelihood or conditional probability $P(x_i/C = k)$ can be estimated with $N_k * i/N_k$, where the numerator is the number of records marked with label k for which the i_{th}-factor is equal to and the denominator is the number of records regarded with label k. The probability $P(X = x)$ is the predictor of the prior probability and is constant with respect to the different classes.
3. **J48**: J48 is the open source Java implementation of the C4.5 algorithm [18] in the weka data mining toolkit. C4.5 builds decision trees from a set of training data in the same way as ID3, using the concept of information gain and entropy. The training data is a set

$$S = s1, s2, \ldots$$

of already classified samples. Each sample consists of a p-dimensional vector

$$(x_{(1,i)}, x_{(2,i)}, \ldots, x_{(p,i)}),$$

where the x_j represent attribute values or features of the sample, as well as the class in which falls. At each node of the tree, C4.5

algorithm selects the attribute of the data that most efficiently splits the samples into subsets enriched in one class or the other. The criterion for splitting is the largest normalized information gain. The attribute with the highest normalized information gain is selected to build the decision. The C4.5 algorithm is then run recursively on smaller sub lists until data are classified [17].

4. **Random Forest**: Random forest [19, 20] is an ensemble learning algorithm for data classification. It is a meta estimator that makes the prediction based on the majority vote of the multitude of decision trees. This classification algorithm reduces the variance of the individual decision trees and makes the classifier more resilient to noise in the training data set. For constructing the Random Forests of m decision trees, m bootstrap samples are generated from the training data set and each of them is utilized for training a decision tree.
5. **Random Tree**: Random decision tree algorithm builds multiple decision trees randomly. While building a decision tree, the classification algorithm picks a "remaining" feature randomly at each node without any accuracy estimation procedure (such as cross validation). A categorical attribute (such as gender) is considered "remaining" if the same categorical attribute has not been selected formerly in a particular decision path arising from the root to the current node of the tree. A continuous attribute (such as income), on the other hand, can be selected more than once in the same decision path. Every time the continuous attribute is selected, a random threshold is chosen.

4. NRFixer: proposed architecture

Objective: To find out if there is a possibility that a bug report currently marked as NR will get fixed in the future.
Input: To find out if there is a possibility that a bug report currently marked as NR will get fixed in the future.
Output: Predicted class: NR-to-fix or NR-to-NR.

Proposed approach: The proposed approach uses bug reports currently marked as NR to predict whether it will be fixed in the future or not. Two prediction models were developed and are compared. To develop the prediction model-1, as shown in Figure 1, it extracts eight bug report meta-fields such as product, component, hardware, severity, priority, cc count, number of comments and keywords to train machine learning classifiers (Zero-R, Naïve Bayes, J48, random forest and random tree). Prediction model-2 additionally uses the extracted parameters namely developer's sentiments and NR bug category along with the eight meta-fields considered in model-1 to train the machine learning classifiers and predicts the class label (NR-to-fix or NR-to-NR).

5. Experimental details

In this section, the experimental details for the prediction of NR-to-fix bugs are presented. For the Eclipse and Firefox projects, various bug report meta-fields were used for prediction and five different classifiers were compared. The classifiers are Zero-R, Naïve Bayes, J48, random forest and random tree. In this experiment a weka toolkit was used. The NR bugs were investigated and the probability of their fixation was predicted.

5.1. Dataset

The data for this study were extracted from the dataset used by Joorabchi et al. [1]. The bug reports of the Firefox and Eclipse projects were used for experimentation. The sentiments of a total of 419 NR bug reports containing 4250 text units were analysed. In the dataset, a single bug report contains a varying number of comment messages which ranges from 1 to 83.

5.2. Experiment parameters

Various meta-fields of bug reports were considered to determine the fixable NR bugs. All factors are listed below:

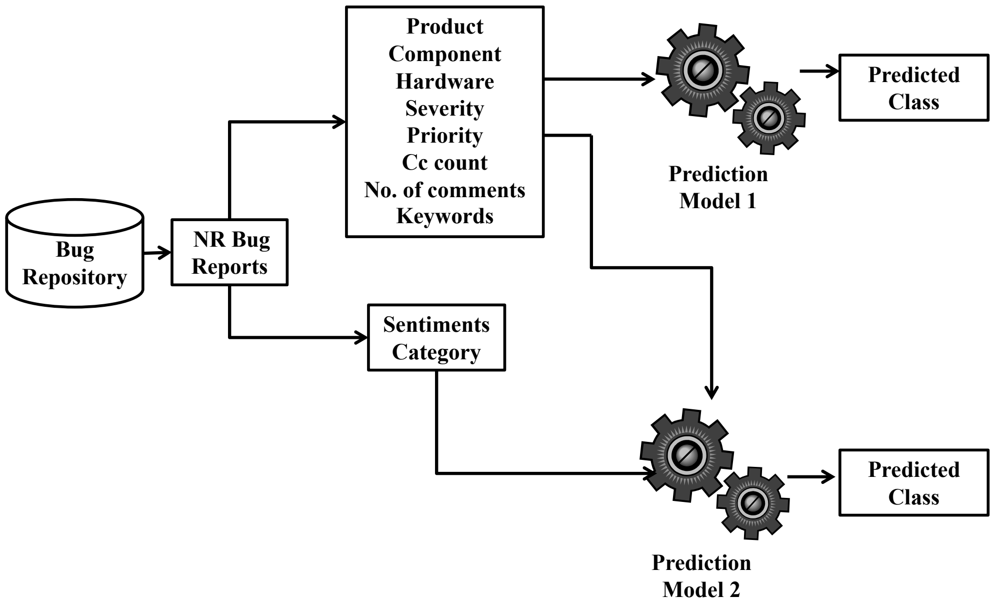

Figure 1. NRFixer: Prediction model for NR bugs

1. **Product**: It refers to the general area the bug belongs to.
2. **Component**: It refers to the second level categorization of a product.
3. **Hardware**: It indicates the computing environment where the bug originated.
4. **Severity**: It describes the impact of the bug. This field offers options, such as severe, normal and minor.
5. **Priority**: The priority field is used to prioritize bug reports. The values of the priority field ranges from P1 to P5 (P1 being the highest priority and P5 being the lowest priority value).
6. **CC Count**: It defines the number of developers in the cc list of the bug report. It is usually the number of developers participating in the bug report.
7. **No. of comments**: It refers to the number of comments made by the developers in order to resolve the bug.
8. **Keywords**: It refers to the tags or categorization of bug reports.

Two more parameters were added in the prediction model:

1. **Sentiment**: It refers to the positive, negative or neutral value expressed by a textual unit.
2. **Category**: It refers to the cause category of the bug. The non-reproducible bugs are classified into six cause categories namely, inter-bug dependencies, environmental differences, insufficient information, conflicting expectations, non-deterministic behaviour and others.

Once the aforementioned factors are extracted, they are used to train various machine learning classifiers in order to predict the fixable bugs from the currently NR marked bugs. The different classifiers used for the prediction of NR in the case of fixed bugs are discussed in the following section.

5.3. Evaluation metrics

To evaluate the performance of the prediction model, standard performance evaluation metrics was used: precision, recall and F-measure.

1. **Precision:** It refers to the fraction of relevant instances retrieved from the total instances that are retrieved.

$$Precision = \frac{tp}{tp + fp}$$

2. **Recall:** It refers to the fraction of relevant instances retrieved from the total relevant instances.

$$Recall = \frac{tp}{tp + fn}$$

3. **F-Measure:** It refers to the harmonic mean of precision and recall.

$$F\text{-}measure = \frac{2 * Precision * Recall}{Precision + Recall}$$

In the equations, tp represents the number of positive samples correctly predicted, fp represents the number of negative samples incorrectly predicted as positive, tn represents the number of positive samples incorrectly predicted and fn represents the number of negative samples correctly predicted. Table 1 shows the confusion matrix.

The tenfold cross validation technique was used in weka to measure the efficiency of NRFixer. When only a limited amount of data is available, cross validation is used to attain an unbiased estimation of model performance. In k-fold cross-validation, data is divided into k subsets of equal size. Thus the model is built k times, each time using sets of data for training the classifier and leaving out one subset as a test set. In order to reduce the impact of the class imbalance problem, the dataset re-sampling technique was used.

6. Experimental results

This section addresses the experimental results of the six identified research questions (RQs). The RQs are classified under two research objectives (RO). RO1 explores the sentiments in bug reports and RO2 investigates the performance of NRFixer.

6.1. Research objective 1 (RO1): exploring sentiments in bug reports

This subsection answers four RQs (RQ1–RQ4) which investigate the sentiments of developers. Joorabchi et al. [1] claimed that a large proportion of bug reports are marked as NR but a small part of these NR bugs (approximately 3%) are fixed later. An empirical evaluation of the sentiments of bug reports was conducted to investigate the developer's behaviour towards R, NR and NR-to-fix bugs. The investigation results of the sentiment analysis of bug reports are presented as below.

RQ1. Do sentiments exist in the NR bug reports? The analysis encompassed 419 NR bug reports to detect whether bug report discussions contain any sentiments or not. These reports constituted a total of 4250 text units (419 taglines, 419 descriptions and 3412 commit comment messages written by software developers). A tagline contains 5 to 10 words and offers a short summary of the bug. A description contains detailed information about the bug and usually constitutes the steps required to reproduce the issue. Commit comment messages contain the developer's discussions regarding the steps that may be useful in bug fixation. The statistics for the sentiment analysis are shown in Figure 2. The percentage of the analysed text units which had either positive or negative sentiments was 65.66%. This confirms the existence of sentiments in software artefacts, such as issue reports as stated by [5].

On the other hand, 34.32% text units exhibited no sentiments. In particular, the tagline field exhibits 67.06% neutral sentiments. This is because a tagline is a 5 to 10-word description of the bug report and thus it is difficult for any sentiment analysis tool to extract the polarity of sentiment in such a small amount of text. Similarly, the description and comments exhibit 39.85% and 29.63% neutral sentiments, respectively. This is due to the fact that these fields sometimes may contain a lot of technical code to resolve the issue and thus may lack any sentiments.

RQ2. What is the difference between sentiments of reproducible bugs and NR bugs? To address this question, two categories of bug reports were considered: the bug reports which are marked as NR at least once in their lifetime and fixed bug reports which are never marked as NR, which are termed as R. For this step, 200 R bug reports were considered (two bug reports from this set contained textual units in languages other than English and thus were removed). Finally 198 R bug reports containing 1556 text units (198 taglines, 198 descriptions and 1160 commit comment messages written by software developers) were studied in addition

Table 1. Confusion Matrix

	Positive (p)	Negative (n)
Positive (p)	*tp*	*fp*
Negative (n)	*fn*	*tn*

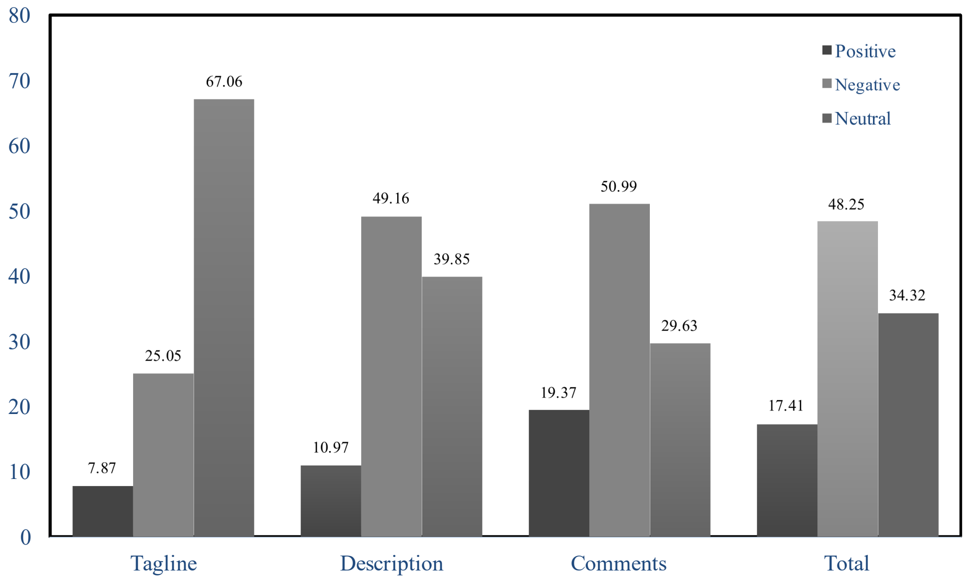

Figure 2. Sentiment distribution in NR bug reports

to 419 NR bug reports (containing 4250 text units). Table 2 presents the statistics of sentiments in reproducible and NR bug reports. The results show that the fraction of negative sentiments is higher in NR bugs (48.25%) than the reproducible bugs (29.24%). Also the fraction of positive sentiments is lower in NR bugs (17.41%) than the reproducible bugs (20.24%). The results confirm that developers have negative sentiments while solving NR bugs. This may be due to the smaller probability of fixing NR bugs.

RQ3. Do the sentiments of developers vary in different categories of NR bugs? To examine this research question, the sentiments of 419 NR bug reports category wise were analysed. The NR bug reports are categorized into six possible cause categories: conflict of knowledge (32 bug reports containing 320 text units), dependent bugs (83 bug reports containing 979 text units), environmental settings (129 bug reports containing 1192 text units), non-deterministic bugs (16 bug reports containing 149 text units), precise information required (113 bug reports containing 1205 text units) and others (46 bug reports containing 405 text units). The statistics for sentiment analysis is shown in Table 3. It was found out that negative sentiments dominated in all six categories of NR bugs. The negative sentiments appear 29.8%–36.24% more than positive sentiments in different cause categories. Among the various categories, the environmental setting contains maximum positive textual units (18.62%) and the category others contains minimum negative textual units (37.17%).

RQ4. Compare the developer's sentiments for the bug report passing through the stages 'Newbug-to-NR' and 'NR-to-fix'? To investigate NR-to-fix bugs, 100 bug reports containing 1648 textual units (100 taglines, 100 descriptions and 1448 textual comments) were considered, they were initially marked from resolution Newbug to NR and were later marked as fixed. NR bugs may go through various stages before being fixed. Stage 1 (Newbug-to-NR) depicts a bug declared as NR and stage 2 (NR-to-Fix) depicts NR being fixed. Figure 3 shows the stages a normal NR bug usually passes through.

Table 2. Percentage distribution of sentiments in various categories of NR bug reports

Bug Reports		Positive (%)	Negative (%)	Neutral (%)
Reproducible	Tagline	5.55	16.66	77.77
	Description	11.11	22.22	66.66
	Comments	24.30	32.59	43.1
	Total	**20.24**	**29.24**	**50.51**
Non-Reproducible	Tagline	7.87	25.05	67.06
	Description	10.97	49.16	39.85
	Comments	19.37	50.99	29.63
	Total	**17.41**	**48.25**	**34.32**

Table 3. Percentage distribution of sentiments in various categories of NR bug reports

Category	Positive (%)	Negative (%)	Neutral (%)
Conflict of Knowledge	16.56	52.18	31.25
Dependant Bugs	15.32	49.23	35.44
Environmental Settings	18.62	51.00	30.36
Non-Deterministic Bugs	16.10	52.34	31.54
Precise Information Required	17.75	47.55	34.68
Others	15.38	37.17	47.43

The investigation was conducted with two different perspectives. For primary investigation, the overall positive and negative percentage of sentiments were searched for in comment messages during both stages. For secondary investigation, the change in sentiments during both stages of each bug report was analysed. To address the developer's sentiments during stage 1 and stage 2, the sentiments of 1448 textual comments were analysed. Table 4 shows the statistics of sentiments at different stages of NR-to-fix bugs.

The primary observation during stage 1 (Newbug-to-NR) comprised 511 textual comments and shows that 13.89% comments have positive sentiments whereas 57.73% comments have negative sentiments. During stage 2 (NR-to-fix), 937 textual comments were analysed. At stage 2, the positive percentage increased to 19.10% whereas negative percentage declined to 47.91%. Thus, it was observed that during stage 2, the positive percentage of sentiments increased by 6% and the negative decreased by 10% as compared to the stage 1. This incline in positivity and decline in negativity reflect the enhanced confidence of developers towards bug reports during the NR-to-fix stage and this optimism leads to fixing NR bugs.

For the secondary investigation, each bug report was analysed to find the change in the positive and negative percentage of sentiments during both stages. It was observed that there is an opinion drift in the sentiments of bug reports in Newbug-to-NR and NR-to-fix stages. The statistics for opinion drift in sentiments during the Newbug-to-NR and NR-to-fix stages is shown in Table 5. This investigation result shows that overall in 71% bug reports either the negativity decreased or positivity increased. It was inferred that during the initial stage of triaging, the developers were reluctant to solve the bug, but this reluctance decreased and as a result NR bugs were fixed.

Investigation summary (RQ1–RQ4). In RO1, four investigations were conducted in the context of the sentiment analysis of bug reports. The investigations confirm that bug reports do express sentiments. The textual units of NR bugs are more inclined towards negative sentiments as compared to reproducible bugs. It has been also found that there is an opinion drift between the Newbug-to-NR and NR-to-fix stages. There

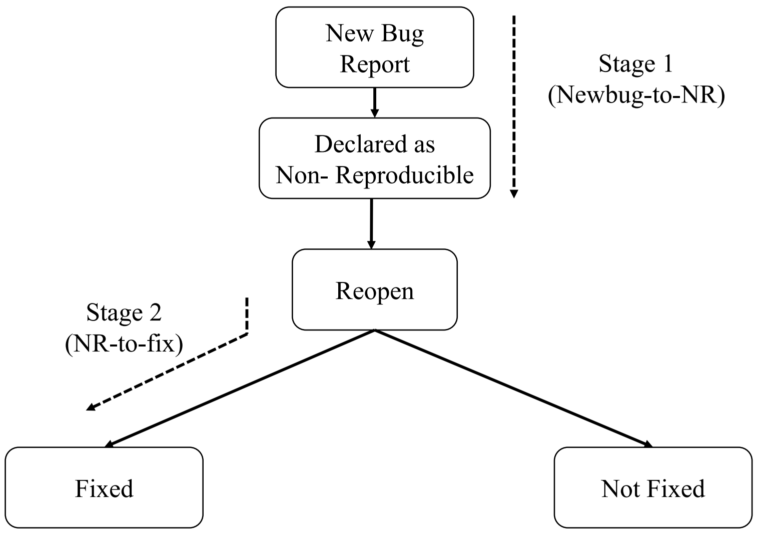

Figure 3. Stages of NR bugs

Table 4. Percentage distribution of sentiments in two different stages of bug reports: Newbug-to-NR and NR-to-fix

Stage	Positive (%)	Negative (%)	Neutral (%)
Newbug-to-NR(Stage 1)	13.89	57.73	28.18
NR-to-fix(Stage 2)	19.10	47.91	32.87

Table 5. Accuracy of NRFixer using various meta-fields of bug report

Opinion drift	% age of bug reports in which		Total % age of bugs
	Positivity increases	Negativity decreases	which observed change
Newbug-to-NR and NR-to-fix (before and after declaring NR)	46%	62%	71%

is a significant drift towards increasing positivity or decreasing negativity in the sentiments of NR bugs during stage NR-to-fix as compared to the Newbug-to-NR stage. This confirms the reluctant behaviour of developers while marking the resolution of a bug report as NR.

The observation of the sentiments of NR bugs highly inclines towards the positive side, it has been inferred that we need an automated way (i.e. prediction model) for judging those NR bugs that have high chances of being fixed. Further, the sentiments can be incorporated to enquire the prediction model's behaviour. This prediction model will not only build the confidence of developers, but will also save time, effort and cost incurred in debugging those bugs which are less likely to be fixed. Thus the NRFixer framework was proposed.

6.2. Research objective 2 (RO2): prediction of the fixability of NR bugs

In this subsection, the performance of NRFixer is investigated (refer to RQ5–RQ6). The results of the research questions addressed in this study are also presented by comparing the performance of five different classifiers: Zero-R, Naïve Bayes, J48, random forest and random tree. In this study, 419 NR bug reports containing 4250 textual units were considered for experimentation. Among these bug reports, 319 bug reports containing 2602 textual units are marked as NR-to-NR (170 bug reports for Mozilla project and 149 bug reports for the eclipse project) whereas 100 bug reports (containing 1648 textual units) are marked as NR-to-fix (50 bug reports

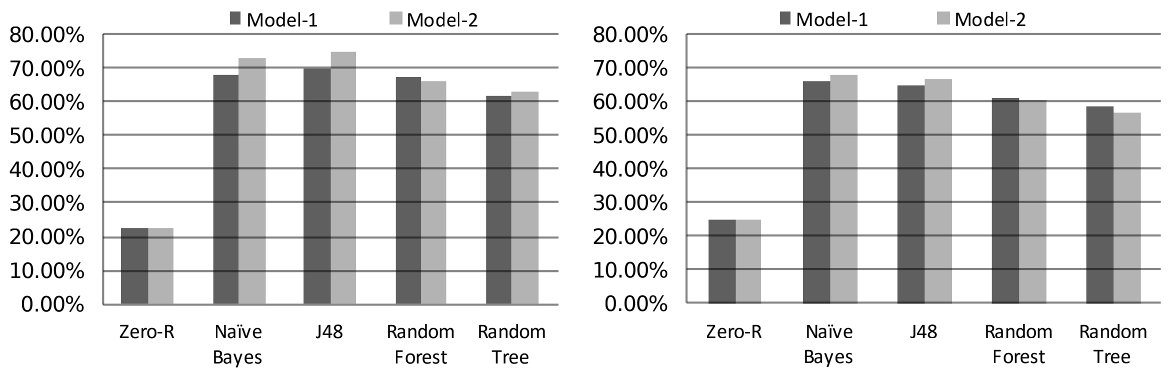

Figure 4. Comparing the accuracy of both models for different classifiers

for the Mozilla project and 50 bug reports for the eclipse project).

RQ5. What is the probability of NR fixing with the use of different classifiers? To address this research question, the performance for prediction model 1 is evaluated. Table 6 presents the precision, recall and F-measure achieved by various machine learning classifiers while using the meta-fields of bug reports, namely product, component, hardware, severity, priority, cc count, number of comments and keywords. For the Firefox project, J48 presents the best precision, recall and F-measure values and the component was found to be the most influencing factor. For the Eclipse project, Naïve Bayes gives the best precision, recall and F-measure values. In this project, severity was found to be the most influencing factor.

RQ6. Does the inclusion of the category of an NR bug and the sentiments of developers affect the efficiency of a prediction model? To address this question, the performance of prediction model-2 in NRFixer is evaluated. Table 7 presents the precision, recall and F-measure achieved by various classifiers using the meta-fields of a bug report, namely product, component, hardware, severity, priority, cc count, number of comments and keywords along with the extracted parameters such as the sentiments of developers and the category of NR bugs. For prediction model-2, the J48 classifier presents the best precision value in the Firefox project and the component was found to be the most influencing factor. Similarly, for the Eclipse project, the Naïve Bayes classifier gives the best precision, recall and F-measure values and severity was found to be the most influencing factor.

The investigation in RO2 confirms that based on bug report meta-fields and sentiment related parameters, it is possible to predict whether the NR bug will get fixed in the future or not. Among different projects, Naïve Bayes and J48 machine learning classifiers achieved the best prediction performance. Taking into consideration the precision metric, J48 is the most suitable classifier for the Firefox project and the Naïve Bayes classifier is the most suitable one for predictions in the Eclipse project. Figure 4 depicts the comparison of accuracy for both prediction models. The inclusion of the sentiments and category of non-reproducible bugs presents better precision of 2–5%.

7. Threats to validity

In this section, we present the various internal and external threats to validity in this work.

External validity. In this work, the bug reports used in experimental evaluation were collected from two popular projects, Firefox and Eclipse of open source bug tracking repository, Bugzilla. Data collected from these projects may vary from other open and closed source projects. Therefore, the outcomes from this study may not generalize well to other commercial software projects. Additional studies are required for other closed source projects or projects that use different software processes. Although we have examined large open source projects which cover a wide range of products, there may be other projects which use different software processes. Thus, the results may not generalize to all of them.

Table 6. Accuracy of NRFixer using various meta-fields of bug report

Project	Classifier	Precision	Recall	F-measure
Firefox	Zero-R	22.7%	22.7%	22.7%
	Naïve Bayes	68%	67.1%	66.6%
	J48	**70.2%**	**70.1%**	**70%**
	Random Forest	67.3%	67.3%	67.3%
	Random Tree	61.8%	61.7%	61.7%
Eclipse	Zero-R	25.2%	25.2%	25.2%
	Naïve Bayes	**66.4%**	**65.2%**	**65.79%**
	J48	65.1%	64.9%	64.99%
	Random Forest	61.3%	61.2%	61.1%
	Random Tree	58.6%	58.6%	58.6%

Table 7. Accuracy of NRFixer using various meta-fields of bug report, sentiments of developers and category of NR bugs

Project	Classifier	Precision	Recall	F-measure
Firefox	Zero-R	22.7%	22.7%	22.7%
	Naïve Bayes	72.9%	**77.5%**	**75.12%**
	J48	**74.7%**	73%	73.84%
	Random Forest	66%	66%	66%
	Random Tree	62.8%	62.5%	62.3%
Eclipse	Zero-R	25.5%	25.2%	25.2%
	Naïve Bayes	**68%**	**65.3%**	**66.62%**
	J48	66.9%	65%	65.6%
	Random Forest	60.5%	60.5%	60.5%
	Random Tree	57%	57%	57%

Internal validity. In this work, it was assumed that the data obtained from a bug repository are optimal. However, there is a possibility of errors or noise in the extracted data, which may affect the results of this study. To mitigate this threat, the bug reports used in this study were obtained from the most widely used projects of the Bugzilla repository. These projects are long lived and are actively maintained, hence it is safe to assume that the extracted data are acceptable (if not optimal). Also, the used dataset suffers from a class imbalance problem and so the re-sampling of dataset was used to overcome this effect. Moreover, the categories of NR bugs may be erroneous. But since the studies related to NR bugs are in their initial phase, this threat was considered to be minor and careful cross-checks of the data and the technique was conducted to eliminate errors in the best possible way.

Five different classifiers were explored: Zero-R, Naïve Bayes, J48, random forest and random tree for the performance evaluation of NRFixer. However, there are many other classifiers (such as genetic algorithms [21], neural network, etc.) and ensemble based techniques [18] (such as stacking, bagging, boosting) which have not been explored in this work. Nevertheless, it is possible that a different set of algorithms would provide better results for NR-to-fix bug prediction as compared to the set of algorithms explored in this work.

Further, in this work, the python NLTK toolkit was utilized [16] for the sentiment analysis of textual contents in bug reports. However, there are many other toolkits available for sentiment analysis, such as SentiStrength, Stanford NLP, etc. Jongeling et al. [22] evaluated various sentiment analysis tools for software engineering studies. They observed that sentiments obtained from various tools neither agree with each other, nor with manual labelling. They suggested a need for a sentiment analysis tool that specifically caters to the software engineering domain. Therefore, the results of this study may improve with the domain specific tailoring of sentiment analysis. But since such a toolkit is not available, we are considering one of the most popular sentiment analysis toolkit, NLTK for experimentation.

8. Conclusion and future work

Non-reproducible bugs are generally frustrating for developers due to the uncertainty of their fixation. To minimize this uncertainty, we have mined the sentiments of textual data present in the non-reproducible bug reports. Mining is done for both categories of bug reports NR-to-fix and NR-to-NR. It was done to find out any clue to assist the developer during the initial stages of bug triaging. The study is being carried out at two different levels. At the first level, the sentiments of bug reports were mined and at the second level framework NRFixer which predicts the probability of NR bug fixation is proposed.

The first level of the study confirms that bug reports do express sentiments. It was found out that the developers possess more negative sentiments for non-reproducible bugs than reproducible bugs. As long as bugs are marked as non-reproducible, the percentage of negative sentiments is 66% bigger than the reproducible bugs. This confirms the reluctant behaviour of developers towards the non-reproducible bugs. It was also found out that there is a major opinion drift found in the sentiments of NR-to-fix bugs. In 71% NR-to-fix bug reports, either there was a decrease in the percentage of negative comments or an increase in the percentage of positive comments when the bug is reopened and is near fixation. This reveals that the developers may have marked the bug as non-reproducible erroneously or it could be due to the lack of some code patch.

The second level of the study deals with the prediction of the possibility of non-reproducible bugs getting fixed using NRFixer. It is evaluated with two dimensions. In the first dimension, we considered various meta-fields of bug report (prediction model-1). In the second dimension, the sentiments of developers were additionally incorporated along with the existing meta-fields of a bug report (prediction model-2). The results of this investigation show that NRFixer could efficiently predict bugs marked as non-reproducible. It was observed that the inclusion of sentiments in prediction model-2 shows an additional rise in the prediction accuracy ranging from 2 to 5% for various classifiers.

For future work, it is planned to explore NRFixer on more machine learning classifiers and software projects, such as closed source applications. Work will also be conducted on improving the performance of NRFixer using ensemble based machine learning techniques. There are also plans to perform a qualitative analysis on domain specific NR-to-fix bug prediction using different textual factors of bug reports. In addition to this work, a fix suggestion tool for non-reproducible bugs that could be fixed will be built.

References

[1] M. Erfani Joorabchi, M. Mirzaaghaei, and A. Mesbah, "Works for me! characterizing non-reproducible bug reports," in *Proceedings of the 11th Working Conference on Mining Software Repositories.* ACM, 2014, pp. 62–71.

[2] E. Murphy-Hill, T. Zimmermann, C. Bird, and N. Nagappan, "The design space of bug fixes and how developers navigate it," *IEEE Transactions on Software Engineering*, Vol. 41, No. 1, 2015, pp. 65–81.

[3] T. Wilson, J. Wiebe, and P. Hoffmann, "Recognizing contextual polarity in phrase-level sentiment analysis," in *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing.* Association for Computational Linguistics, 2005, pp. 347–354.

[4] F. Jurado and P. Rodriguez, "Sentiment analysis in monitoring software development processes: An exploratory case study on GitHub's project issues," *Journal of Systems and Software*, Vol. 104, 2015, pp. 82–89.

[5] A. Murgia, P. Tourani, B. Adams, and M. Ortu, "Do developers feel emotions? an exploratory

analysis of emotions in software artifacts," in *Proceedings of the 11th working conference on mining software repositories*. ACM, 2014, pp. 262–271.

[6] P. Tourani, Y. Jiang, and B. Adams, "Monitoring sentiment in open source mailing lists: Exploratory study on the apache ecosystem," in *Proceedings of 24th Annual International Conference on Computer Science and Software Engineering*. IBM Corp., 2014, pp. 34–44.

[7] D. Garcia, M.S. Zanetti, and F. Schweitzer, "The role of emotions in contributors activity: A case study on the Gentoo community," in *The Third International Conference on Cloud and Green Computing (CGC)*. IEEE, 2013, pp. 410–417.

[8] D. Pletea, B. Vasilescu, and A. Serebrenik, "Security and emotion: Sentiment analysis of security discussions on GitHub," in *Proceedings of the 11th working conference on mining software repositories*. ACM, 2014, pp. 348–351.

[9] E. Guzman, D. Azócar, and Y. Li, "Sentiment analysis of commit comments in GitHub: An empirical study," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 352–355.

[10] G. Destefanis, M. Ortu, S. Counsell, S. Swift, M. Marchesi, and R. Tonelli, "Software development: Do good manners matter?" *PeerJ Computer Science*, Vol. 2, 2016, p. e73.

[11] H. Valdivia Garcia and E. Shihab, "Characterizing and predicting blocking bugs in open source projects," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 72–81.

[12] E. Shihab, A. Ihara, Y. Kamei, W.M. Ibrahim, M. Ohira, B. Adams, A.E. Hassan, and K. Matsumoto, "Studying re-opened bugs in open source software," *Empirical Software Engineering*, Vol. 18, No. 5, 2013, pp. 1005–1042.

[13] R. Hewett and P. Kijsanayothin, "On modeling software defect repair time," *Empirical Software Engineering*, Vol. 14, No. 2, 2009, p. 165.

[14] P.J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, "Characterizing and predicting which bugs get fixed: An empirical study of Microsoft Windows," in *ACM/IEEE 32nd International Conference on Software Engineering*, Vol. 1. IEEE, 2010, pp. 495–504.

[15] T. Zimmermann, N. Nagappan, P.J. Guo, and B. Murphy, "Characterizing and predicting which bugs get reopened," in *Proceedings of the 34th International Conference on Software Engineering*. IEEE Press, 2012, pp. 1074–1083.

[16] Python NLTK sentiment analysis with text classification demo. [Online]. http://text-processing.com/demo/sentiment/ [Accessed September 2016].

[17] A. Padhye, Classification methods. [Online]. http://www.d.umn.edu/~padhy005/Chapter5.html [Accessed September 2016].

[18] J.R. Quinlan, *C4.5: programs for machine learning*. Elsevier, 2014.

[19] L. Breiman, "Random forests," *Machine learning*, Vol. 45, No. 1, 2001, pp. 5–32.

[20] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques*. Elsevier, 2011.

[21] M. Mitchell, *An introduction to genetic algorithms*. MIT press, 1998.

[22] R. Jongeling, S. Datta, and A. Serebrenik, "Choosing your weapons: On sentiment analysis tools for software engineering research," in *IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2015, pp. 531–535.