Empirical Software Engineering (2022) 27: 92
https://doi.org/10.1007/s10664-022-10133-6

Revisiting reopened bugs in open source software systems

Ankur Tagra[1] · Haoxiang Zhang[2] · Gopi Krishnan Rajbahadur[2] · Ahmed E. Hassan[1]

Accepted: 16 February 2022 / Published online: 25 April 2022

Abstract

Reopened bugs can degrade the overall quality of a software system since they require unnecessary rework by developers. Moreover, reopened bugs also lead to a loss of trust in the end-users regarding the quality of the software. Thus, predicting bugs that might be reopened could be extremely helpful for software developers to avoid rework. Prior studies on reopened bug prediction focus only on three open source projects (i.e., Apache, Eclipse, and OpenOffice) to generate insights. We observe that one out of the three projects (i.e., Apache) has a data leak issue – the bug status of *reopened* was included as training data to predict reopened bugs. In addition, prior studies used an outdated prediction model pipeline (i.e., with old techniques for constructing a prediction model) to predict reopened bugs. Therefore, we revisit the reopened bugs study on a large scale dataset consisting of 47 projects tracked by JIRA using the modern techniques such as SMOTE, permutation importance together with 7 different machine learning models. We study the reopened bugs using a mixed methods approach (i.e., both quantitative and qualitative study). We find that: 1) After using an updated reopened bug prediction model pipeline, only 34% projects give an acceptable performance with AUC $\geqslant 0.7$. 2) There are four major reasons for a bug getting reopened, that is, technical (i.e., patch/integration issues), documentation, human (i.e., due to incorrect bug assessment), and reasons not shown in the bug reports. 3) In projects with an acceptable AUC, 94% of the reopened bugs are due to patch issues (i.e., the usage of an incorrect patch) identified before bug reopening. Our study revisits reopened bugs and provides new insights into developer's bug reopening activities.

Communicated by: Andy Zaidman

✉ Haoxiang Zhang
haoxiang.zhang@acm.org

Ankur Tagra
atagra@cs.queensu.ca

Gopi Krishnan Rajbahadur
gopi.krishnan.rajbahadur1@huawei.com

Ahmed E. Hassan
ahmed@cs.queensu.ca

[1] Software Analysis and Intelligence Lab (SAIL), Queen's University, Kingston, ON, Canada

[2] Centre for Software Excellence at Huawei, Kingston, ON, Canada

Keywords Bug reports · Reopened bugs · Data quality · Open source software · Model interpretation

1 Introduction

Fixing bugs is an important part of software development. Generally, a bug is resolved and closed after the bug is fixed. However, sometimes a bug needs to be reopened. Mi and Keung (2016) observed that around 6–10% of the bugs are reopened in four open source projects from the Eclipse product family. If a fair number of fixed bugs get reopened, it is an indication of instability in the software system (Zimmermann et al. 2012). Reopened bugs consume more time to resolve than normal bugs by a factor of 1.6–2.1 (Mi and Keung 2016). Reopened bugs cause extra rework effort for development teams (Mi et al. 2018). Moreover, reopened bugs also lead to a loss of trust in the end-users regarding the quality of the software (Shihab et al. 2010). Figure 1 shows a reopened bug[1]. In this issue, the bug was *fixed* and *closed*; however, later another developer reopened the bug and fixed it. It took the bug an extra 42 days from initial *close* to the bug getting *reopened* and finally *closed*.

Therefore, to avoid the reopening of a bug, prior studies proposed models to predict if a bug will be reopened (i.e., reopened bug prediction models). For example, Shihab et al. (2013) studied the reopening likelihood of a bug based on four dimensions of features (i.e. work habits, bug report, bug fix, and team dimension). They studied three projects (i.e. Apache, Eclipse, and OpenOffice) using 44,317 non-reopened bugs and 11,745 reopened bugs. Xia et al. (2015) proposed the ReopenPredictor[2] and further improved the performance of the reopened bug prediction model using the same dataset as used by Shihab et al. Shihab et al. (2013). Xia et al. Xia et al. (2015) found that the model to predict reopened bugs show a consistent performance on the three studied projects. We examined the bug reports from the data shared by Shihab et al. (2013) and Xia et al. (2015), we observed that one out of the three projects (i.e., Apache project) has a data leak issue (Kaufman et al. 2012; Tu et al. 2018). Several instances of the feature *last status* contained the value of reopened (not all the instances of Apache project). This suggests that the dataset to train the prediction model of reopened bugs contained the information that a bug was reopened already. When the data from the future is used to predict current events, it amounts to data leak (Kaufman et al. 2012). Therefore, if we discount the findings from the Apache project, the findings of the prior studies are effectively based on only two projects (i.e., Eclipse, and OpenOffice). Furthermore, all these prior studies used an old prediction pipeline and with time, the techniques for prediction models have improved. For instance, no prior study in reopened bug prediction tuned the hyperparameters of their studied model, which could, in turn, improve the prediction performance of the constructed reopened bug prediction model (Rajbahadur et al. 2019; Tantithamthavorn et al. 2018). No prior study in reopened bug prediction performed correlation and redundancy analysis on the independent features of their dataset to remove correlated features; however, several recent studies have used correlation and redundancy analysis on their dataset prior to training the prediction model (Jiarpakdee et al. 2019; Tantithamthavorn et al. 2018; Rajbahadur et al. 2019, 2017; Lee et al. 2020; da Costa et al. 2018; McIntosh et al. 2016). This motivated us to do a large scale study of the

[1] https://issues.apache.org/jira/browse/XW-140
[2] https://github.com/xin-xia/reopenBug

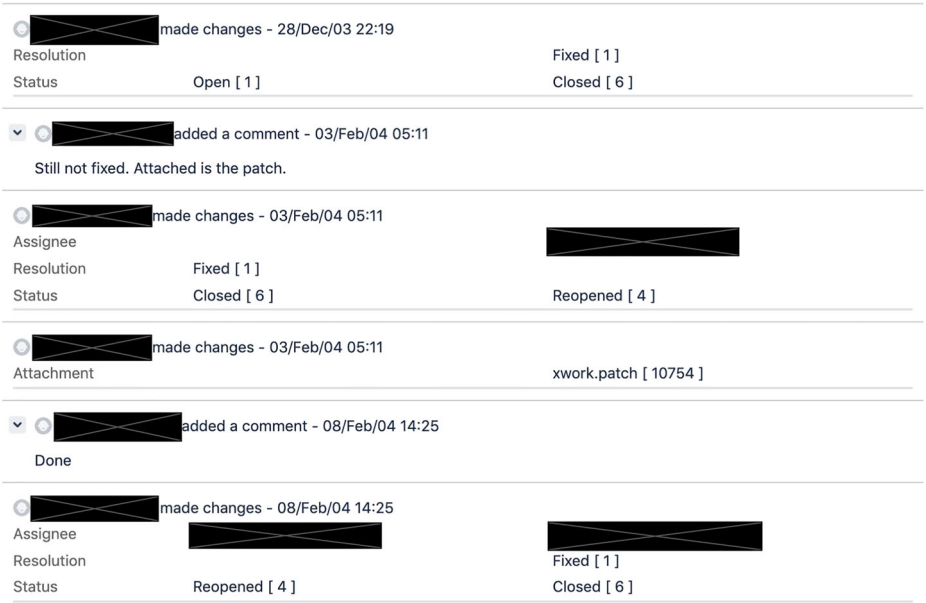

Fig. 1 Historical events in a reopened bug

likelihood of reopening a bug using the latest prediction pipeline. In our study, we answer the following research questions.

RQ1: How well do bug reopen models generalize? ***Motivation:*** Prior studies on reopened bugs focus only on three projects (Apache, Eclipse, OpenOffice) (Shihab et al. 2013; Xia et al. 2015). Out of these three projects Apache project has data leak issues. In addition, Several components used for the prediction of reopened bugs (such as the strategy for handling class imbalance, generating feature importance of the prediction model) have shown to be not as effective (e.g., the strategy for handling class imbalance, generating feature importance of the prediction model). Therefore, in this RQ, we revisit the prior findings of reopened bug prediction on a large scale study of 47 new projects that we collect from JIRA as we outline in Section 2 and with the latest techniques for building reopened bug prediction pipeline to determine the likelihood of bug reopening.

Results: Using our updated reopened bug prediction model pipeline, we observe that across the 47 studied projects, at best we are able to achieve an acceptable AUC ($\geqslant 0.7$) on 34% (16/47) of the projects. Furthermore, reopened bug prediction models constructed with Random Forest and Gradient Boosting models yield an acceptable AUC ($\geqslant 0.7$) on more studied projects than the model constructed with other classifiers (for example, Decision Tree and Adaboost). Finally, with our large scale study, we observe that *number of comments* is the most important feature that helps in predicting the reopening likelihood of a bug. Such a finding contrasts with the findings observed by Shihab et al. Shihab et al. (2013) where they found that the *comment text* is the most important feature in determining the likelihood of reopening a bug.

RQ2: Why do bugs get reopened? ***Motivation:*** (Zimmermann et al. 2012) observe that understanding bug reopening is required since it helps in various software development

tasks such as identifying areas which need better tool support and improving the bug triage process. We observe by interpreting the reopened bug prediction model that *number of comments* and *comment text* are the top two most important features in determining the likelihood of reopening a bug. These insights are coarse and do not provide enough actionable suggestions that can be used to either prevent bug reopening or identify the potential root cause of reopened bugs. Therefore, in this RQ, we investigate the rationale behind why bugs are reopened by leveraging the whole history of bug reports. In addition, we investigate if the reasons for reopening a bug differ between the projects on which the constructed reopened bug prediction models have an acceptable and a poor performance.

Results: Through a mixed method approach, we identify that in 63% and 86% of the reopened bugs, developers post comments during and after the reopening respectively, and in 77% of the reopened bugs, the reason to reopen a bug is identified either during or after the bug reopening. Through a qualitative study of 370 samples of reopened bugs (with a 95% confidence level and a 5% confidence interval), we identify four categories of reasons for reopening a bug, that is, technical (i.e., patch/integration issues), documentation (i.e., updating fields of a bug report), human (i.e., incorrect assessment by a developer), and no stated reason in a bug report. In addition, through our qualitative study, we observe that in projects with an acceptable AUC, 94% of the reopened bugs are due to patch issues and a negligible proportion of 4% bugs are reopened due to integration issues, whereas in projects with a poor AUC, apart from patch issues (66%), bugs are reopened due to integration issues (22%).

Based on our findings, we encourage developers to carefully examine their patches for bug fixing, e.g., by testing the patches thoroughly before resolving the bugs, as the most common reasons for reopening bugs are due to technical issues (i.e., patch/integration issues). Also, we suggest that the bug tracking systems should enable developers to edit certain fields of bug reports without requiring the reopening of a bug, as 24% of our manually studied reopened bugs are reopened due to documentation (bookkeeping) issues.

The rest of the paper is organized as follows. Section 2 describes the data collection processes. Section 3 discusses our case study. Section 4 presents the implications of our findings. Section 5 discusses the related work. Section 6 provides the threats to the validity of our study. Finally, Section 7 discusses the conclusion of our study.

2 Data Collection and Preprocessing

We describe our data collection process in this section. For revisiting the reopened bugs study on a large dataset, we collect 47 projects from JIRA (i.e., JIRA dataset) and map all the features of the Bugzilla dataset that are also present in the JIRA dataset. In Section 2.1, we describe the data collection processes of the JIRA dataset. The JIRA dataset contains categorical (i.e., nominal), numeric, boolean, and text features. We discuss these features in detail in Section 2.1, and in Section 2.2 we discuss our process to clean the data for the reopened bug models.

2.1 Collecting the JIRA dataset

We study the reopened bugs across all public projects that are tracked by JIRA and whose source code is hosted on Github. JIRA is a commonly used bug tracking system and it

is custom tailored to support 3,000 apps[3]. We use the JIRA REST API[4] to extract JIRA bugs. We use query subfield *type=bug* to ensure that we only collect the bugs from JIRA. Firstly, we collected all the bugs that were ever reopened across all public projects tracked by Apache JIRA. For non-reopened bugs, we collected all the bugs which were either closed or resolved but were never reopened. In total, we collected 388,404 non-reopened bugs and 25,376 reopened bugs across 589 projects. All the features except the *number of files in the fix* were collected using the JIRA REST API. We used Pydriller[5] to collect the number of files changed in the corresponding GitHub repositories using the approach outlined by Vieira et al. (2019). Table 1 shows the 20 features that we collect for our study. Note that the features (i.e., *platform*, *severity*, *severity changed* and *number in the CC list*) are not present in the JIRA. We further filter the studied projects using two criteria: 1) we select the top 50 projects in JIRA based on the committers count[6] since they are quite representative of popular Apache projects, 2) We remove the projects that do not have at least 10 reopened bugs. We do so, as in Section 3.1, we discuss that, in our model we use 10-times-stratified 10-fold cross validation. When we 10-fold split a dataset with fewer than 10 samples of positive class (reopened bugs), there will be at least one fold with no positive samples. Thus the model will not get trained for that fold (unavailability of positive sample). By applying these two criteria, we end up with 47 projects to study (including 178,636 non-reopened bugs and 9,993 reopened bugs). We provide the replication package[7] with data of the 47 projects in our study. While computing the value of features, we ignore all events in the bug report that occur at the time of reopening the bug or after reopening the bug, to prevent any data leak issue. For example, for the *comment text* feature, we consider all the comments before the first reopening of each studied bug. We do not consider comments of subsequent reopens in a bug report. A bug can be reopened multiple times; however, in this study we only study the prediction of the first reopening of a bug by leveraging activities before the first reopening of a bug.

2.2 Pre-processing features

Figure 2 shows the steps we followed to collect bug reports and how we process the data. Our dataset contains two text features, i.e., the *description text* and *comment text*. We pre-processed the text features similar to prior studies (Uysal and Gunal 2014; Denny and Spirling 2017; Kannan and Gurusamy 2014). The comments and description of a bug report contain URLs. We replace them with a placeholder token to simplify the representation of URLs (Hemalatha et al. 2012). Developers add code blocks in the description or comments of a bug report. To simplify our text pre-processing, we replace code blocks with a place-holder token. Error logs and stack traces are processed similarly, i.e., by replacing them with a placeholder token. We then remove all the stop words as they do not contribute much to the context of reopened bugs (Song et al. 2005). We then use the NLTK package[8] to stem the text to reduce the different forms of a word (Srividhya and Anitha 2010; Méndez et al. 2005; Hardeniya et al. 2016). Finally, we use the pre-processed text in our reopened bug

[3] https://www.atlassian.com/software/jira
[4] https://developer.atlassian.com/cloud/jira/platform/rest/v2/intro/
[5] https://pydriller.readthedocs.io/en/latest/
[6] https://projects.apache.org/projects.html?number
[7] https://zenodo.org/record/6378876#.YjrqlxDMJQI
[8] https://www.nltk.org/

Table 1 The features that we use in our reopened bug prediction model

Category	Features used by Shihab et al.	Our features (JIRA)	Type	Explanation
Work habits	Time	Time	Nominal	Time of the day when first closing the bug (e.g., morning)
	Weekday	Weekday	Nominal	Day of the week when first closing the bug (e.g., monday)
	Month day	Month day	Numeric	Day of the month when first closing the bug
	Month	Month	Numeric	Month when first closing the bug
	Day of year	Day of year	Numeric	Day of the year when first closing the bug
Bug report	Component	Component	Nominal	The component of the project that the bug belong to
	Platform	-	-	-
	Severity	-	-	-
	Priority	Priority	Numeric	The priority of the bug
	Number in the CC list	-	-	-
	Description size	Description size	Numeric	The number of words in the description
	Description text	Description text	Text	The text of the description of a bug report
	Number of comments	Number of comments	Numeric	The number of comments in a bug report
	Comment size	Comment size	Numeric	The number of words in the comments
	Comment text	Comment text	Text	The text of the comments from a bug report
	Priority changed	Priority changed	Boolean	Whether the priority level changed in a bug report
	Severity changed	-	-	-
Bug fix	Time days	Time days	Numeric	The time it took to close the bug
	Last status	Last status	Nominal	The last status when the bug was first closed
	Number of files in the fix	Number of files in the fix	Numeric	The number of files in the bug fix
People	Reporter name	Reporter name	Nominal	The name of the reporter in the bug report
	Fixer name	Assignee name	Nominal	The name of the assignee in the bug report

Table 1 (continued)

Category	Features used by Shihab et al.	Our features (JIRA)	Type	Explanation
	Reporter experience	Reporter experience	Numeric	The number of bugs that a reporter has reported before reporting this bug
	Fixer experience	Assignee experience	Numeric	The number of bugs assigned to the assignee before this bug

prediction model together with the other features that are categorical (e.g., *component*, *last status*, and *assignee*), numeric (e.g., *month*, *description size*, and *number of comments*), or boolean (e.g., *priority changed*).

3 Case Study

In this section, we describe our case study results based on our new dataset to study reopened bugs. We first use the new JIRA dataset to predict reopened bugs (in Section 3.1). Then

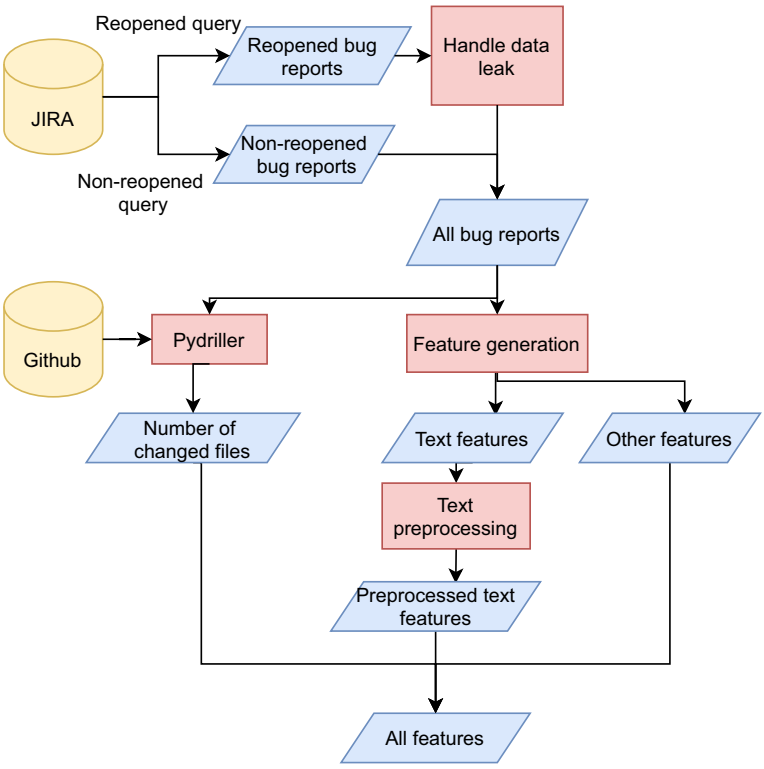

Fig. 2 Workflow of data collection and pre-processing

we analyze the studied bugs to understand the rationale for reopening a bug report (in Section 3.2). For each RQ, we provide the motivation, approach, and results of the RQ.

3.1 RQ1: How well do bug reopen models generalize?

Motivation: Prior studies on determining the likelihood of reopening a bug by Shihab et al. (2013) and later by (Xia et al. 2015) were based only on three projects (i.e., Apache, Eclipse, and OpenOffice). Moreover, the other two studies by Shihab et al. (2010) and by Xia et al. (2013) on reopened bug prediction were based only on one project (i.e., Eclipse). After examining the dataset used by these prior studies, we observe that out of their studied three projects, one project (i.e., Apache) has a data leak issue – the bug status of *reopened* was included as training data to predict reopened bugs. Hence, prior findings on determining the likelihood of reopening a bug were effectively based only on two projects. In addition, these prior studies were conducted 6–10 years ago. Several experimental components that the prior studies used to build a reopened bug prediction model pipeline have since then been shown to be not as effective when building a prediction model. For example, prior studies on reopened bug prediction did not tune the hyperparameters of their used model. However, several recent studies have shown that tuning the hyperparameters of a model is pivotal to ensure its optimal performance and interpretation (Fu et al. 2016; Tantithamthavorn et al. 2016a, 2018). Furthermore, prior studies (Shihab et al. 2013) used a class re-weighting and re-sampling strategy to deal with the class imbalance issue that is present in the projects. However, several recent studies have shown SMOTE (Synthetic Minority Over-sampling Technique) to be a more effective method for class re-balancing (Tantithamthavorn et al. 2018; Agrawal and Menzies 2018). Prior studies (Shihab et al. 2013) used top node analysis with a Decision Tree model for generating feature importance of their model; however, recent studies show that the results of top node analysis can be biased since Decision Tree favors categorical features (Altmann et al. 2010). Moreover, prior studies (Shihab et al. 2013; Xia et al. 2015) did not use feature encoding with the latest and more widely used encoding techniques (Cerda et al. 2018) such as one-hot encoding to encode the categorical features.

After Shihab et al. (2013) and Xia et al.'s (2015) study, no subsequent study revisited the reopened bug prediction problem with these pipeline improvements in mind. Therefore, in this RQ, we revisit their findings on a large scale study of 47 new projects that we collect from JIRA as we outline in Section 2 and with the latest techniques for building reopened bug prediction model pipeline to determine the likelihood of bug reopening.

Approach: We construct our reopened bug prediction model in this RQ. Figure 3 shows the prediction model pipeline that we use to construct our reopened bug prediction model. We discuss each step of the pipeline in detail below.

Step 1: Correlation and redundancy analysis. Tantithamthavorn et al. (2018) outlines that presence of correlated and redundant features might result in affecting the computed feature importance ranks of a model. Therefore, similar to prior studies (Jiarpakdee et al. 2019; Tantithamthavorn et al. 2018; Rajbahadur et al. 2019, 2017; Lee et al. 2020; da Costa et al. 2018; McIntosh et al. 2016), we perform a correlation and redundancy analysis on the independent features of our dataset to remove correlated features using the implementation provided by AutoSpearman method of

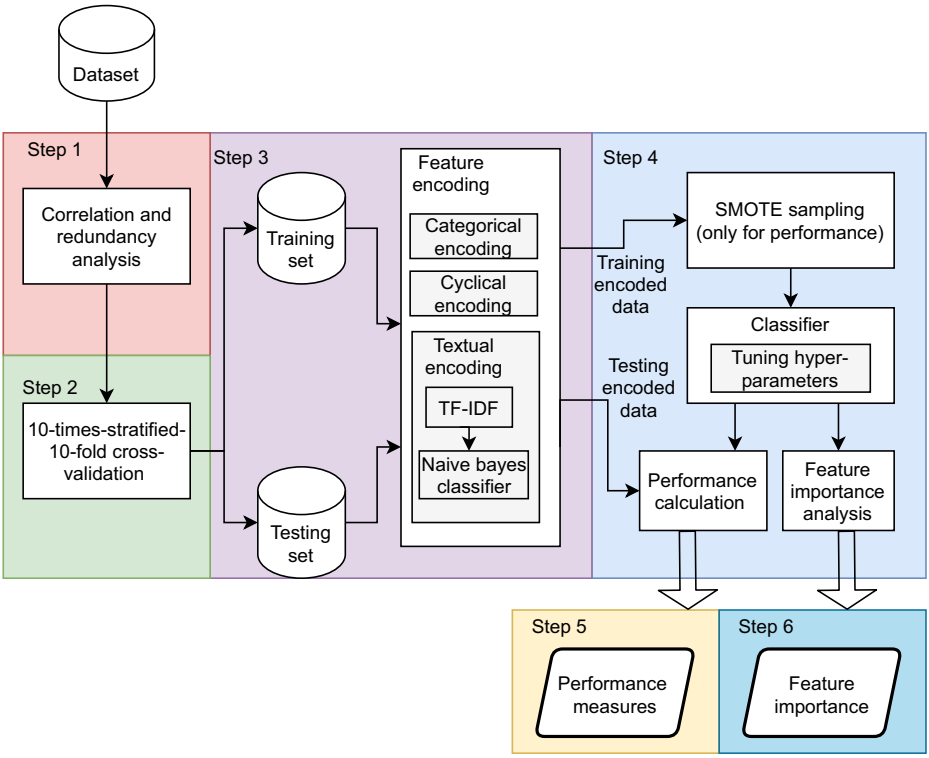

Fig. 3 Reopened bug prediction model pipeline

Rnalytica[9] R package (Tantithamthavorn et al. 2018; Lee et al. 2020; Yatish et al. 2019). We choose the AutoSpearman method to remove the correlated and redundant features as the study by Jiarpakdee et al. (2019) showed that selecting features with AutoSpearman typically yields better subset of features than other techniques. In addition, Jirapakdee et al. also show that when using AutoSpearman, the impact of the feature selection on the performance of the trained model is minimal. Finally, AutoSpearman is a method is commonly used in prior studies (Jiarpakdee et al. 2019; Lee et al. 2020; Rajbahadur et al. 2021). Autospearman uses a criteria that selects one feature from a group of highest correlated features which shares the least correlation with other features that are not in the group (Jiarpakdee et al. 2018) based on Spearman correlation score. Following which, AutoSpearman then removes the variables which have VIF>=5. We use a default value of ρ for the Spearman correlation coefficient threshold to remove the correlated features.

Step 2: Splitting dataset into training/testing sets. We split the dataset into training and testing sets using 10-times-stratified 10-fold cross-validation as opposed to 10-times 10-fold cross-validation used by Shihab et al. (2013). In stratified 10-fold cross-validation, the whole dataset is split into 10 folds in such a way that each fold contains the same percentage of all class labels as in the overall dataset. Each fold is then used to test the model performance and the remaining 9 folds are used for

[9] https://rdrr.io/github/software-analytics/Rnalytica/man/AutoSpearman.html

training the model. We repeat this process 10 times. Therefore, for each iteration 10 performance and feature importance measures are generated and an overall 100 performance and feature importance measures are generated on each studied project. For more details of the stratified k-fold cross-validation, please refer to Zeng et al. (2019). We choose 10-times-stratified-10-fold cross-validation in particular because our studied dataset is imbalanced, and stratified k-fold cross-validation helps preserve the original class distribution of the data (He and Ma 2013; Arellano 2019).

Step 3: Feature encoding.

Step 3.1 Categorical features encoding. We encode the categorical features in the dataset using one-hot feature encoding to transform categorical features present in the dataset into numeric features to be used as an input to train the model (Zheng and Casari 2018). We employ one-hot feature encoding as opposed to simply converting the categorical features to numeric features as Cerda et al. (2018) stated that one-hot feature encoding is one of the latest and more widely used categorical feature encoding procedure. One-hot encoding creates a separate boolean column for each category of the categorical features. These columns are then passed along with the other numeric columns to the next step.

Step 3.2 Cyclical features encoding. For the feature *weekday* we use cyclical encoding as feature *weekday* values are cyclic in nature (Hébert 2020). We do so as we need to maintain unit distance between each pair of consecutive days for cyclic values (Chakraborty and Elzarka 2019).

Step 3.3 Textual features encoding. We use the TF-IDF (term frequency-inverse document frequency) vectorization (Nyamawe et al. 2020) to encode the two text features (i.e., *comment text* and *description text*) that are present in the studied datasets similar to several prior studies (Biggers et al. 2014; McMillan et al. 2011; Rakha et al. 2017; Nyamawe et al. 2020). TF-IDF works by assigning a higher weight to a word if that word appears many times within a small number of documents. Similarly, TF-IDF assigns lower weight to a word, when the given word appears fewer times in a document. TF-IDF assigns the lowest weight to a word if that given word appears in almost all the documents (Corazza et al. 2016). We use the training data in each iteration to build our TF-IDF model and use the trained TF-IDF model to generate the TF-IDF scores for text features in the testing data.

After transforming the text features with TF-IDF, we encode the text features as probabilities with a Naive Bayes model similar to the prior study (Shihab et al. 2013; Meyer and Whateley 2004; Zhang et al. 2020). We follow the above approach since using only TF-IDF creates a column in the dataset for each word present in the dataset. Such an encoding might drastically increase the number of studied features and cause our reopened bug prediction models to overfit the studied datasets. Furthermore, when we interpret the constructed reopened bug prediction models, we are more interested in observing the importance of the studied text features (i.e., the *comment text* and *description text*) in predicting if a bug would be reopened (rather than the individual words). Therefore, we train two Naive Bayes models on the training data.

Step 4: Model construction.

Step 4.1 Handling class imbalance for the reopened bug prediction model. We show the distribution of percentage of reopened bugs in all projects in Fig. 4. Since the dataset for determining the likelihood of reopening a bug is imbalanced (i.e., fewer reopened bugs as compared to non-reopened bugs), we used the SMOTE technique to rebalance the training dataset (Malhotra and Khanna 2017). SMOTE is an

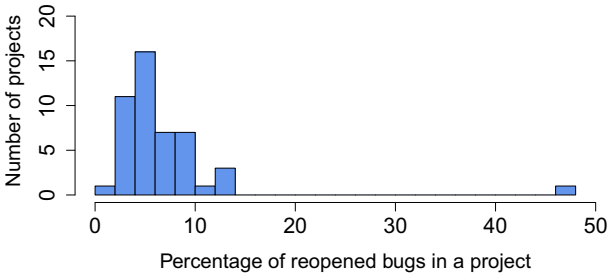

Fig. 4 Percentage of reopened bugs in all project

oversampling approach in which the minority class is over-sampled by generating synthetic samples (Chawla et al. 2002). We use default values such as *auto* and 5 for parameters sampling strategy and k_neighbors respectively used in SMOTE provided by library imblearn[10]. *Sampling strategy* is set based on input data type. If *auto* input is provided, it automatically detects the input type and sets the *sampling strategy* accordingly. Shihab et al. (2013) used a class re-weighting and re-sampling strategy to deal with the class imbalance present in the dataset. However, we choose SMOTE for handling class imbalance as recent studies show that SMOTE is a more effective method for class re-balancing (Tantithamthavorn et al. 2018; Agrawal and Menzies 2018).

Step 4.2 Training the model. We use the training set to train the reopened bug prediction model. In our analysis, we use 7 commonly used models in software analytics as mentioned in Table 2. We choose these models in particular as a recent study by Agrawal et al. (2019) notes, Decision Tree, Random Forest, Logistic Regression, Multinomial Naive Bayes and K-nearest neighbors (KNN) are five of the most commonly used models in software analytics. In addition, we also include Gradient Boosting and Adaboost models as they are also commonly used in several software analytics studies (Ghotra et al. 2015; Tantithamthavorn et al. 2016a, 2018).

Step 4.3 Tuning the hyperparameters of the model. We use a grid search (Scikit-learn 2020) based hyperparameters tuning to optimize the performance of the model. Grid search exhaustively considers all combinations of parameter values used to train the model and chooses the parameter that yields the best performance (we use AUC) for the constructed reopened bug prediction model. Shihab et al. Shihab et al. (2013) used a Decision Tree model without tuning the hyperparameters to build their reopened bug prediction model. However, several recent studies have shown that tuning the hyperparameters of a model is pivotal to ensure its optimal performance and interpretation (Fu et al. 2016; Tantithamthavorn et al. 2016a, 2018).

Step 5: Model performance evaluation. In this step, we measure the performance of various models on determining the likelihood of reopening a bug. We choose the AUC measure because the AUC measure is both threshold-independent and class imbalance insensitive. Several prior studies recommend the usage of AUC performance measure to evaluate the performance of the models in software analytics studies (Tantithamthavorn et al. 2016a; Rajbahadur et al. 2017, 2019; Ghotra et al. 2017; Lessmann et al. 2008). We find the performance of the reopened bug prediction

[10] https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html

Table 2 Overview of the used hyperparameters in our study

Model	Parameter name	Parameter description	Parameter range
Random Forest	n_estimators	The number of trees in the forest	(**100**, intranduniform(50, 150))
	criterion	The function for measuring the split quality	(**gini**, entropy)
	min_samples _split	The minimum needed samples to split a node	(**2**, randuniform(0.0, 1.0))
Gradient Boosting	n_estimators	The number of boosting stages to perform	(**100**, intranduniform(50, 150))
	min_samples _leaf	The minimum needed samples to be at a leaf node	(**1**, 1, 10))
Logistic Regression	penalty	To specify the used norm in the penalization	(l1,**l2**)
	tol	The tolerance for the stopping criteria	(**1e-4**, randuniform(0.0,0.1))
	C	The inverse of regularization strength	(**1**, intranduniform(1,500))
Adaboost	n_estimators	The maximum number of estimators at which the boosting is terminated	(**50**, intranduniform(50, 150))
Multinomial Naive Bayes	alpha	The additive smoothing parameter	(**1**, randuniform(0.0, 1.0))
Decision Tree	splitter	Choosing strategy for splitting at each node	(**best**, random)
	criterion	The function for measuring the split quality	(**gini**, entropy)
	min_samples _split	The minimum needed samples to split a node	(**2**, randuniform(0.0,1.0))[a]
KNN	n_neighbors	The number of neighbors	(**5**, intranduniform(2, 25))
	weights	The weight function used in prediction	(**uniform**, distance)
	p	Power parameter	(**2**, intranduniform(1,15))
	metric	The distance metric to use for the tree	(**minkowski**, chebyshev)

[a]https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

The default values are shown in bold. Randuniform denotes drawing random samples from a uniform distribution, and intranduniform denotes denotes drawing random samples from a uniform distribution and converting them to their nearest integer value

model on a given project to be acceptable if on the given project mean AUC value of the constructed reopened bug prediction model across the 100 iterations is greater than 0.7 (Jiarpakdee et al. 2020; Morasca and Lavazza 2020; Al Dallal and Morasca 2014; Turabieh et al. 2019). Otherwise, we deem the performance of the reopened bug prediction model as poor.

Step 6: Ranking important features. To generate the feature importance in predicting the reopened bugs, we use the permutation feature importance method (Altmann et al. 2010). We use the permutation feature importance method over the top node analysis used by Shihab et al. (2013) for the following reasons. First, Shihab et al. compute the feature importance ranks on re-balanced datasets. Second, the results of top node analysis can be biased since Decision Tree favors categorical features (Altmann et al. 2010). To avoid these shortcomings, we use a permutation feature importance method. The permutation feature importance method works as follows. First, we compute the performance of reopened bug prediction model. We then randomly permute the values of each feature at a time and compute the model's performance to note how much performance drop does permuting a feature encounters when compared to the model built on all the features whose values are not permuted. A larger drop in performance due to a permutation of a feature's value signifies the higher importance of that feature.

For computing the feature importance ranks, we use the dataset as it is to train the model (i.e., we do not use SMOTE in feature importance analysis). Tantithamthavorn et al. (2018) find that using class re-balancing techniques when computing feature importance ranks results may lead to wrong features being considered as the important features. We compute the feature importance ranks only on the projects on which the best performing reopened bug prediction models have an acceptable AUC as (Lipton 2018; Chen et al. 2018) argue that for a model to be interpreted, it needs to have an acceptable performance. After this step for each iteration of training data, feature importance scores are obtained for each feature. More details of the working of permutation feature importance method can be found in (Altmann et al. 2010). After generating the feature importance score, we compute the features importance ranks using Scott-Knott ESD test (Tantithamthavorn et al. 2016b). These ranks determine the order of importance of features in predicting the reopened bugs. We finally note the most common top two most important features used by the best performing model across all the studied projects.

Results: Across the 47 studied projects, at best, we observe an acceptable AUC ($\geqslant$ 0.7) for 34% (16/47) of the projects. Table 3 shows the percentage of projects on which our constructed reopened bug prediction models deliver an acceptable AUC. We observe that on 66% projects we get a poor AUC ($<$ 0.7) for predicting reopened bugs. In the study by Shihab et al. (2013), they found that the prediction models have a very high performance within the three studied projects. However, our finding suggests that the reopened bug prediction model does not necessarily show a very high performance on all projects when testing with a larger collection of projects.

Reopened bug prediction model pipeline constructed with Random Forest and Gradient Boosting models yield an acceptable performance on more studied projects (AUC $\geqslant$ 0.7) than a pipeline with other models. From Table 3, we observe that the Random Forest model and Gradient Boosting model yield an acceptable AUC on 34% of the studied projects. Whereas, the Decision Tree and KNN models give an acceptable AUC on only 10% of the projects. Such a result argues that future studies should consider Random Forest and Gradient Boosting models to construct reopened bug prediction models.

For the projects with an acceptable AUC, the *number of comments* is the most important feature. Such a result is in contrast with the findings by Shihab et al. (2013) who found that *comment text* is the most important feature in determining the likelihood of reopening a bug. We observe that using the best performing model (i.e., Random Forest), the *number of*

Table 3 Percentage of projects with which the constructed reopened bug prediction model pipeline with different models yields an acceptable AUC

Model	Projects with acceptable AUC (AUC$\geqslant$ 0.7) count (mean AUC)	Projects with poor AUC (i.e., AUC$<$ 0.7) count (mean AUC)	Percentage of the acceptable projects
Random Forest	16 (0.78)	31 (0.63)	**34%**
Gradient Boosting	16 (0.77)	31 (0.63)	**34%**
Logistic Regression	14 (0.78)	33 (0.63)	29.8%
Adaboost	12 (0.79)	35 (0.63)	25.5%
Multinomial Naive Bayes	8 (0.75)	39 (0.61)	17%
Decision Tree	5 (0.82)	42 (0.59)	10.6%
KNN	5 (0.76)	42 (0.58)	10.6%

comments is the most important feature to predict reopened bugs in 31% of the projects with an acceptable AUC. We further find that the *comment text* which Shihab et al. found to be the most important feature in 66% (i.e., two out of three) projects, is the most important feature in only 12% projects and second most important feature in 31% projects. Future studies can leverage the number of comments as a parameter in selection criteria for reopened bugs study.

Summary of RQ1

Using our updated reopened bug prediction model pipeline onto 47 JIRA projects, we find that only on 34% of the studied projects we are able to achieve an acceptable AUC ($\geqslant 0.7$) for our constructed reopened bug prediction model. Furthermore, we observe that the Random Forest model and Gradient Boosting model give the best AUC in determining the likelihood of reopening a bug. Finally, we also observe that *number of comments* is the most important feature in determining the likelihood of reopening a bug.

3.2 RQ2: Why do bugs get reopened?

Motivation: The reopened bug prediction model interpretation generates coarse insights in prior studies. For example, by interpreting the reopened bug prediction model, we could determine that *number of comments* and *comment text* are the top two most important features in determining the likelihood of reopening a bug; however, such insights do not provide any actionable suggestions that can be used to either prevent bug reopening or identify the root cause of reopening bugs. Moreover, we observe that some developers pointed out the rationale to reopen the bug during reopening the bug. For example, Fig. 5 shows a bug report[11] (ID: 4112) for the Apache Accumulo project. In this bug report, the bug was considered as non-reproducible initially and resolved but later a developer reopened the bug and at the same time commented about the reason to reopen the bug. However, comments

[11] https://issues.apache.org/jira/browse/ACCUMULO-4112

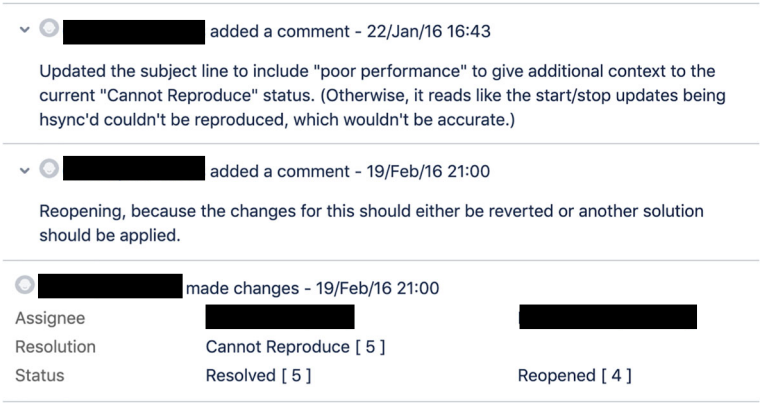

Fig. 5 A bug report (ID: 4112) for the Apache Accumulo project, showing a comment depicting reason for reopening the bug at the same time when the bug is reopened, hence this comment is not used in reopened bug prediction model

which are provided either during bug reopening or after reopening a bug are not used in the reopened bug prediction model in order to prevent data leak. This useful data (i.e., events during and post reopening bugs) can be analyzed using other approaches (such as a qualitative analysis) to generate insights into reopened bugs. Therefore, in this RQ, we investigate what are the rationale to reopen bugs. Moreover, since we observe in Section 3.1 that only 34% of the studied projects give an acceptable AUC for predicting reopened bugs, in this RQ, we also investigate if the reasons for reopening a bug differ between the projects on which the constructed reopened bug prediction models have an acceptable and a poor performance. In order to better understand the rationale for reopening bugs, we leverage the whole history of bug reports to understand the characteristics of reopened bugs. We wish to gain insights into the challenges in building reopened bug prediction models for better understanding and managing reopened bugs.

Approach: We used all the 9,993 reopened bugs from 47 projects in our analysis. A mixed methods approach was used (including both the quantitative and qualitative study approach) in our study. We initially performed a quantitative analysis to identify the co-occurring changes at the time of reopening the bug. We then performed a qualitative analysis to identify the reasons why bugs get reopened.

Quantitative study: The aim of the quantitative study is to identify what are all the changes that occur in reopened bugs. For our quantitative study, we divided all the changes that occur in the bug reports into three categories (i.e., before reopening, during reopening, after reopening). We considered all such events that occur before reopening a bug as the *before reopening* category and all such events that occur at the time of reopening a bug as the *during reopening* category. We considered all such events that occur after a bug is reopened as the *after reopening* category. We extracted all the fields of changes during reopening a bug and identify the commonly occurring change patterns.

Qualitative study: For our qualitative study, we first randomly selected reopened bugs, we then conducted an open coding study (Corbin and Strauss 1990) to understand the reasons that bugs get reopened.

There are 9,993 reopened bugs from the 47 projects on JIRA. Out of these bugs, we selected a random sample of 370 reopened bugs. Our selection reaches a statistically representative sample of all the reopened bugs associated with the 47 projects with a 95% confidence level and a 5% confidence interval (Boslaugh 2012; Zhang et al. 2019). For our qualitative study, the first three authors of the paper examined all the comments associated with our manually studied bugs, i.e., comments before/during/after a bug is reopened. For example, in a bug report[12] (ID: 34059) for the Apache Flex project, a developer commented in 40 minutes after the bug was reopened that: "*because the issue also happens with TextInputSkin (see ML piotr.zarzycki)*."

Three coders (i.e., the first three authors) conducted the manual study of tagging the reasons to reopen bugs. We aimed to identify two labels, i.e., what is the reason to reopen a bug, and whether the rationale for reopening the bug occurred before, during, or after the bug is reopened? We discuss the steps in the open coding process below:

Step 1: Round-1 coding. We randomly selected 50 bugs out of the 370 reopened bugs. We tagged these bugs individually. Each coder's tagging process was independent. We did not interfere in other coder's tagging process.

Step 2: Discussion after Round-1 coding. After the first round, we obtained around 23 tags. These tags are specific reasons to reopen a bug. We then discussed to cluster these tags into categories of tags. After this meeting, we finalized the tag categories for the Round-1 coding.

Step 3: Revisit Round-1 coding. After the discussion, we re-coded the same 50 bugs with the updated tag categories.

Step 4: Round-2 coding. The remaining 320 bugs were used for the Round-2 coding. We assigned each coder with two-thirds of the remaining bugs (i.e., 214 bugs), such that each bug was tagged exactly by two coders. We were allowed to add new tags if a bug could not be classified by the existing Round-1 tag categories. Round-2 coding was also done individually by each coder without the influence of other coders. It took eight hours for each one of the coders to finish Round-2 coding.

Step 5: Discussion after Round-2 coding. After the Round-2 coding, we discussed the results and resolved any disagreement. After this meeting, we finalized the tags for our study.

Step 6: Revisiting Round-1 and Round-2 coding. After the discussion in the previous round, we finally revisited all the bugs of Round-1 and Round-2 coding to update the new tags obtained after Round-2 coding. This step took several hours for each coder. This step was performed without the interference of other coders so as to maintain the independence of decision while tagging bugs.

Step 7: Computing inter-coder agreement. We used the tags to compute the inter-coder agreement. The inter-coder agreement is a measure to compute the agreement reached in tagging a bug. A bug reopening reason is considered as a match if both the coders provide the same tag for a bug and a mismatch if both the coders provide different tags for a bug. We computed the inter-coder agreement using Krippendorff's α (Artstein and Poesio 2008; Hayes and Krippendorff 2007). The inter-coder agreement for our study is 0.87 which concludes that our qualitative analysis is reliable as prior studies use $\alpha \geqslant 0.8$ as an indicator of reliable agreement (Li et al. 2020; Beckler et al. 2018; Webb et al. 2020; Vassallo et al. 2020; Scoccia and Autili 2020).

[12] https://issues.apache.org/jira/browse/FLEX-34059

Step 8: Dealing with disagreements. After coding all the bugs, we discussed the disagreements among the coders for bugs with different tags. If a bug was tagged differently by two coders, and they mutually did not agree to a common tag, then the tag of the bug was determined by the third coder.

For understanding the rationale for reopening a bug, we used four predefined categories (i.e., before reopening, during reopening, after reopening, and no evidence found) as our tags. Each coder tagged the evidence for the reopened bugs that are assigned to him when labeling the reasons.

After completing the above qualitative study, we studied the distribution of reasons for reopening bugs in projects with an acceptable AUC and projects with a poor AUC. To determine the projects that have an acceptable AUC and a poor AUC in the reopened bug prediction model, we considered the findings from Section 3.1. For this study, we classified the projects as acceptable and poor based on the best performing model (i.e., Random forest) identified in Section 3.1. We labelled all the 370 manually studied reopened bugs based on the project category as *reopened bug from projects with acceptable AUC* and *reopened bug from projects with poor AUC*. After labeling these 370 manually studied reopened bugs, we observed that 80 reopened bugs have a project category tag as *reopened bug from projects with acceptable AUC* and the remaining 290 reopened bugs have a project category tag as *reopened bug from projects with poor AUC*. To compare the distribution of reasons to reopen a bug in projects with acceptable and poor AUC, we needed to use the same dataset as we used to train the reopened bug prediction model. In our reopened bug prediction model we only used the dataset before reopening the bug. Therefore, to compare the distributions of reasons to reopen bugs in projects with acceptable and poor AUC, we only used those bugs where the reason to reopen a bug was identified before reopening the bug. From our first qualitative study, we had obtained the evidence tags for reopened bugs with four predefined categories (i.e., before reopening, during reopening, after reopening, and no evidence found). Using these *evidence* tags and *project category* tags, we identified that 13 out of 80 reopened bugs with project category as *reopened bug* have *evidence* tag as *before reopening*, and 44 out of 290 reopened bugs with project category as *reopened bug* have *evidence* tag as *before reopening*. Note that in this analysis, we only computed and trusted the insights from projects with an acceptable AUC. We did so, as prior work (Lipton 2018) states that insights from only projects with an acceptable performance can be used. Therefore initially, we considered only 13 reopened bugs in projects with acceptable AUC and 44 reopened bugs in projects with poor AUC. However, this sample of reopened bugs was too small for any meaningful conclusion from the study, therefore, we added more samples in each project category (i.e., *reopened bug from acceptable project* and *reopened bug from poor project*) of reopened bugs, until we obtained 50 reopened bugs with evidence tag as *before reopening* from projects with acceptable AUC and 50 reopened bugs with *evidence* tag as *before reopening* from projects with poor AUC.

Results: Reopened bugs have an average number of 2.4 other changes of the bug report during bug reopening. Apart from the obvious change of reverting the resolution (98%) and posting a comment (63%) at the time of reopening a bug, 12% of our studied bugs are reassigned during reopening. From a qualitative study, we observe that in 77% of the reopened bugs, the rationale for reopening is provided either during or after the reopening event to support why a bug was reopened. On the other hand, in 15% of the reopened bugs, the rationale is provided before reopening, and in 7% the rationale is not provided Using our quantitative study, we identify all the field changes that occur in reopened bug reports.

Table 4 The distribution of major field changes that occur in reopened bugs grouped by before, during, and after reopening a bug

Change field	Before	During	After
Resolution	9,632 (96.4%)	9,810 (98.2%)	8,227 (82.3%)
Comment	9,371 (93.8%)	6,301 (63.1%)	8,666 (86.7%)
Assignee	4,956 (49.6%)	1,216 (12.2%)	1,916 (19.2%)
Fix version	5,053 (50.6%)	1,072 (10.7%)	4,962 (49.7%)
Attachment	3,750 (37.5%)	211 (2.1%)	2,104 (21.1%)
Link	2,144 (21.5%)	199 (2%)	1,796 (18%)

Table 4 shows the distribution of field changes in the reopened bugs in different time periods (i.e., before reopening, during reopening, and after reopening). While analyzing the reopened bugs, we observe that 19% of the reopened bugs are reassigned at least once after being reopened. Yadav et al. (2019) observe that reassigning bugs during bug fixing is not always harmful since bug reassignment can involve knowledge transfer among developers. In our study, we observe that in 2% (226 out of 9,993) of the reopened bugs, developers change the assignee of the bug and resolve/close the bug immediately, indicating that the bug was reopened solely for the purpose of updating the assignee field of the bug report; however, the bug was already fixed. For example, in a bug report[13] (ID: 741) for the Apache Airvata project, the bug assignee was changed while reopening the bug and the bug was immediately closed. In 9% (990 out of 9,993) of the reopened bugs, the developers change the assignee and do not immediately close/resolve the bug, suggesting that the bug was not fixed and re-assigned to another developer during reopening. For example, in a bug report[14] (ID: 4640) for the Apache Accumulo project, the bug was re-assigned at the time of reopening the bug; however, the new assignee worked on the bug later and eventually resolved the bug. Prior studies observe that with an increasing number of bug reassignments, time to fix the bug also increases (Guo et al. 2011; Saha et al. 2015). The bug resolution process (i.e., during bug reopening and after bug reopening) is rich and contains many insights about bug reopening activities; however, no prior study has used this dataset to generate insights into bug reopening.

71% (i.e., 263 out of 370) of the manually studied reopened bugs, the rationale for reopening is provided at the time of reopening the bug. Table 5 shows the rationale with their bug count and proportion. For example, in a bug report[15] (ID: 2197) for the Apache OFBiz project, the developer posted a comment about reopening the bug at the time of reopening the bug: "*Reopened issue to re-examine the latest patch of CSS changes to the bluelight theme.*" In another example, a bug report[16] (ID: 11429) for the Apache Ignite project was reopened to update the resolution from *fixed* to *duplicate*, which occurred at the same time as reopening the bug. In 6% (i.e., 24 out of 370) of the studied reopened bugs, the rationale is provided after the bug is reopened. For example, in a bug report[17] (ID: 3598) for the Apache NetBeans project, after reopening the bug the developer discussed that the patch did not work.

[13] https://issues.apache.org/jira/browse/AIRAVATA-741
[14] https://issues.apache.org/jira/browse/ACCUMULO-4640
[15] https://issues.apache.org/jira/browse/OFBIZ-2197
[16] https://issues.apache.org/jira/browse/IGNITE-11429
[17] https://issues.apache.org/jira/browse/NETBEANS-3598

Table 5 Different reasons for reopening the bugs based on various rationale categories (i.e., before bug reopening, during bug reopening and after bug reopening

Category	Reason	Explanation	Example	# projects	Overall count (%)	Before reopening count (%)	During reopening count (%)	After reopening count (%)
Technical	Patch	Bug is not fixed by the patch.	Reopening, since it seems this issue is causing problems with high load. Will revert and test.[a]	37	202 (54.6%)	50 (13.5%)	142 (38.4%)	10 (2.7%)
	Integration	Bug is reopened because of an issue during the patch integration process.	This has been reverted, it breaks unit tests.[b]					
Documentation	Update	Bug is reopened to update a field.	reopen to update version[c]	31	90 (24.3%)	3 (0.8%)	78 (21.1%)	9 (2.4%)
Human	Incorrect assessment	Bug is reopened due to incorrect assessment.	not a duplicate. Same stack trace, but root cause is different[d]	28	52 (14.1%)	4 (1.1%)	43 (11.6%)	5 (1.4%)
No reason	No stated reason	No reason was found in the bug report for reopening.	Not A Problem[e]	15	26 (7%)	-	-	-
Total				47	370 (100%)	57 (15.4%)	263 (71.1%)	24 (6.5%)

[a] https://issues.apache.org/jira/browse/HBASE-14689

[b] https://issues.apache.org/jira/browse/AMBARI-14383

[c] https://issues.apache.org/jira/browse/HAWQ-227

[d] https://issues.apache.org/jira/browse/HADOOP-10589

[e] https://issues.apache.org/jira/browse/STORM-190

Although we observed in Section 3.1, that *comment text* is among the most important features in determining the likelihood of reopening a bug, many rationale in the history of bug reports can't be leveraged in determining the likelihood of reopening a bug as they occur mainly at the time of bug reopening or even after a bug was reopened. However, we encourage future research to leverage this set of data (i.e., during and after bug reopening) to alleviate the reopened bug management.

We identify four categories of reasons for reopening bugs, including technical (i.e., patch/integration issues), documentation (bookkeeping), human, and no stated reason. Table 5 shows the reason categories together with their number/proportion, explanation, and examples. We discuss the reasons for reopening bugs below:

- **Inline with common intuition, the majority (i.e., 54%) of our studied bugs are reopened due to technical issues during the bug fixing processes, that is, the failure of code patch and patch integration.** More specifically, 41% (i.e., 154 out of 370) of the reopened bugs are due to patch issues. Some of the major patch issues include runtime errors, regression, code improvements, and code merge issues. For example, a bug report[18] (ID: 5206) for the Apache Flink project is reopened due to runtime exception. Patch issues can deprecate the quality of the system and hence need to be fixed before resolving the bug. However, the above bug was not carefully tested before it was marked as fixed. During bug reopening, a developer pointed out that there is a runtime exception issue in the system and it took around 1.3 years to fix the reopened bug. In addition, 13% (i.e., 48 out of 370) of our studied reopened bugs are due to integration failure. Integration issues include build failures, tests failure, flaky tests[19], and unit test failures. For example, in a bug report[20] (ID: 346) for the Apache Daffodil project, test cases failure leads to the bug reopening. In this bug, a developer had already warned to test the fix before closing the issue; however, after closing the bug, a developer found that test cases are failing leading to reopening the bug. It took three days extra to resolve the issue, from the time the test cases failure issue was first observed. Stolberg (2009) suggested certain code integration testing techniques such as running all unit tests with every build and developing unit tests for all new code that allow fixing bugs at a cheaper cost. We suggest the developer community to follow such code integration testing techniques during bug fixing process in order to avoid reopening a bug later due to integration issues.
- **In 24% (i.e., 90 out of 370) of the studied reopened bugs, bugs are reopened due to bookkeeping issues (i.e., updating a field in the bug report).** Bug reports have various bug fields, such as *version*, *assignee*, and *resolution*. In order for developers to update these bug fields, the bug needs to be reopened. We observe that 10% (i.e., 39 out of 370) bugs are reopened to update the fix version. For example, a bug report[21] (ID: 480) for the Apache HAWQ project was reopened just to update its associated version. Moreover, 7% (i.e., 26 out of 370) bugs are reopened to update the resolution. For example, a bug report[22] (ID: 1999) for the Apache Felix project was reopened and the developer who reopened the bug posted a comment to point out that it was reopened to fix the resolution. 1% (i.e., 5 out of 370) bugs are reopened to update the

[18] https://issues.apache.org/jira/browse/FLINK-5206
[19] https://whatis.techtarget.com/definition/flaky-test
[20] https://issues.apache.org/jira/browse/DAFFODIL-346
[21] https://issues.apache.org/jira/browse/HAWQ-480
[22] https://issues.apache.org/jira/browse/FELIX-1999

assignee. For example, in a bug report[23] (ID: 3598) for the Apache Spark project, the bug was reopened to change its assignee. Bugs that are reopened to update the meta information of a fixed bug can use other mechanisms to avoid any confusion with a reopened bug due to technical issues. So far, technical issues and documentation issues are not differentiated by prior studies in determining the likelihood of reopening a bug. New bug features can be used to better identify reopened bugs due to technical issues instead of documentation issues.

- **In 14% (i.e., 52 out of 370) of our studied reopened bugs, bugs are reopened due to an incorrect initial assessment.** These bugs include cases where a bug was wrongly assessed initially, leading to the reopening of the bug at a later stage. We observe that 3% (i.e., 14 out of 370) of our studied bugs were reopened since they were wrongly classified as *non-reproducible*. In this case, if a developer is not able to reproduce the bug, then he stops working on the bug and resolves the bug without fixing it by considering the bug as *non-reproducible*. For example, in a bug report[24] (ID: 2570) for the Apache Thrift project, a developer initially closed the bug by mentioning "*can't reproduce ... feel free to re-open if new information becomes available*", but after almost a month, another developer reopened the bug and fixed the bug. To handle such issues, the reporter of the bug should provide detailed reproduction steps for assignees to work on the bug properly. 2% (i.e., 10 out of 370) of our studied bugs were wrongly considered as a duplicate. For example, a bug[25] (ID: 17024) for the Apache Spark project is considered as a duplicate initially; however, after 8 days, another developer reopened the bug and started working on fixing the bug. Prior work (Jalbert and Weimer 2008) also studied duplicate bug reports and found that in some projects almost a quarter of all bugs are duplicate.
- **In 7% (i.e., 26 out of 370) of our studied reopened bugs, we were not able to identify the reason for reopening the bug.** In these bugs, developers do not provide any rationale in the bug to identify the reason for reopening the bug. For example, in a bug report[26] (ID: 1152) for the Apache HAWQ project, no comment is provided at the time of reopening the bug.

In addition, we compute the closed to reopened time and reopened to fixed time for all categories of reopened bugs, and compute whether these reopened bugs are finally fixed or not. Table 6 shows that for reopened bugs in the technical and human category, the median time from reopened to fixed is much longer than the median time from closed to reopened.

94% and 4% of the total bugs in projects with an acceptable AUC are reopened due to patch issues and integration issues respectively. However, in projects with a poor AUC, only 66% and 22% bugs are reopened due to patch issues and integration issues respectively. Table 7 shows the distribution of reason categories identified before reopening the bug together with their number/proportion for both projects with acceptable as well as poor AUC. In the projects where we get an acceptable AUC, we manually classify the bugs with reasons before the reopening event. We observe that in 50 out of 330 randomly studied reopened bugs from projects with acceptable AUC, the reason to reopen bugs was identified before bug reopening. Out of these 50 bugs, there are 49 (98%) bugs where the reason

[23] https://issues.apache.org/jira/browse/SPARK-3598
[24] https://issues.apache.org/jira/browse/THRIFT-2570
[25] https://issues.apache.org/jira/browse/SPARK-17024
[26] https://issues.apache.org/jira/browse/HAWQ-1152

Table 6 Distribution of the median time spent in various states of reopened bugs

Category	Reason	# of bugs finally fixed (% bugs)	Median time from closed to reopened (in days)	Median time from reopened to fixed (in days)
Technical	Patch Integration	190 (94.1%)	1.5	16
Documentation	Update	87 (96.7%)	2	0
Human	Incorrect assessment	47 (90.4%)	0.7	17.7
No reason	No stated reason	26 (100%)	0	0

to reopen a bug was identified before bug reopening are reopened due to technical issues; however, no bug is identified where the reason identified before bug reopening is due to documentation issues. For example, in a bug report[27] (ID: 2626) for the Apache Cassandra project, the developer comments before reopening the bug that *"Thanks for your quick response. I'm not sure but I think that the fix doesn't work. Types of pendingFlush, sstables and compacting objects are java.util.Collections.UnmodifiableSet. Those are not instances of org.apache.commons.collections.set.UnmodifiableSet."* stating that the bug was reopened due to technical reasons. In projects with an acceptable AUC, a technical reopening reason can be identified before reopening events, suggesting that new features that can characterize technical issues can be used in a reopened bug prediction model. Moreover, we observe that in 50 out of 313 randomly studied reopened bugs from projects with poor AUC reason to reopen a bug was identified before bug reopening. Out of these 50 bugs, 11 (22%) are reopened due to integration issues, this is higher than the amount of bugs reopened due to integration issues (i.e., 4%) identified before bug reopening in projects with an acceptable performance. Apart from bugs reopening in projects with poor performance due to technical issues, 3 (6%) bugs where the reason to reopen a bug was identified before bug reopening are reopened due to documentation issues. For example, in a bug report[28] (ID: 15519) for the Apache Spark project, a developer comments *"I misclicked and resolved as Fixed instead of as Duplicate. Feel free to edit."* Since bug reopening due to documentation issues is a type of trivial bookkeeping activity, we encourage future studies to remove the bug reopens due to documentation issues from their study of determining the likelihood of reopening a bug.

From the study using the before reopening bug dataset, we observe that 98% and 88% of bugs in projects with an acceptable and a poor performance respectively are reopened due to technical issues. All these reasons are identified by developer comments, suggesting that *comment text* is an important feature in the study of reopened bugs, which supports our finding in RQ1. From the study using during/after bug reopening dataset and before bug reopening dataset, we observe that in 77% of the bugs the reason to reopen a bug is identified during/after reopening the bug, this is 5 times more than bug reopening reasons identified before (i.e., 15%) bug reopening, indicating that 5 times more data can be leveraged while considering during/after bug reopening dataset in future bug reopening studies. Moreover, it also signifies that useful discussions (i.e., comments about bug reopening reasons) occur more frequently during/after bug reopening than before bug reopening.

[27] https://issues.apache.org/jira/browse/CASSANDRA-2626
[28] https://issues.apache.org/jira/browse/SPARK-15519

Table 7 Different reasons for reopening the bugs identified before bug reopening in various project categories (i.e., acceptable, and poor)

Category	Reason	Evidence time	Acceptable performance projects	# Extra reopened bugs needed in acceptable projects	Poor performance projects	# Extra reopened bugs needed in poor projects
Technical	Patch	before	47(94%)	250	33 (66%)	23
	Integration	before	2 (4%)	250	11 (22%)	23
Documentation	Update	before	0 (0%)	250	3 (6%)	23
Human	Incorrect assessment	before	1 (2%)	250	3 (6%)	23
No reason	No stated reason	before	-	250	-	23

Summary of RQ2

In 77% of the reopened bugs, the reason to reopen a bug can be identified either during or after a bug is reopened; however, this rich dataset (i.e., during and after bug reopening) is never studied specifically. 54% of our manually studied reopened bugs are due to technical issues (i.e., the initial patching/integration process fails). 24% of our manually studied reopened bugs are due to documentation issues (i.e., updating a field of the bug). 14% of our manually studied reopened bugs are due to human related issues (i.e., incorrect assessment). In projects with an acceptable AUC, 94%, and 4% of the bugs where the reason to reopen a bug is identified before bug reopening are due to patch issues and integration issues respectively, while in projects with a poor AUC, apart from patch issues (i.e., 66%), bugs are reopened due to integration issues (i.e., 22%).

4 The implications of our findings

Table 8 shows the findings of our study and their implications. We discuss implications specific to each interest group (i.e., developers, and researchers). Note that, we think that these implications can be of more interest to a particular target group.

4.1 Implications for developers:

Developers should use code integration testing techniques such as running all unit tests with builds before resolving bugs in order to avoid bug reopening later due to integration issues. We observe in RQ2 (Section 3.2) that 13% (i.e., 48 out of 370) of the manually studied reopened bugs are due to the integration issues. Moreover, we also observe in RQ2 that 22% of the bugs in projects with poor AUC are reopened due to integration issues identified before bug reopening. These issues include build failure, tests failure, unit tests failure, flaky tests, and tests timed out. Some of these issues can be handled during the bug fixing process, such as by executing unit tests locally and only committing the patch when the unit tests pass. Developers can also use automated tools for verifying the test accuracy. Stolberg (2009) suggested certain code integration testing techniques such as running all unit tests with every build and developing unit tests for all new code that allow fixing bugs at a cheaper cost. Developers can use such techniques to resolve the bugs in order to avoid bugs getting reopened due to integration issues. Other issues, particularly the build success can also be tested after committing the patch. However, developers may be careless in performing such tests, or simply skip them to directly resolve the bugs. Additionally, we observe that there is a varying trend among development teams in waiting for the build to succeed and then resolving bugs. For example, in a bug report[29] (ID: 4531) for the Apache Thrift project, the developer waits for the build to succeed then resolves the bug. Whereas, in a bug report[30] (ID: 15977) for the Apache HBase project, the developer resolves the bug immediately

[29] https://issues.apache.org/jira/browse/THRIFT-4531
[30] https://issues.apache.org/jira/browse/HBASE-15977

Table 8 Our major findings and their implications

Audience	Findings	Implications
Developers	13% (i.e., 48 out of 370) of the manually studied bugs are reopened due to the integration issues and 22% (11 out of 50) of the bugs in projects with poor AUC are reopened due to integration issues.	Developers should use code integration testing techniques such as running all unit tests with builds before resolving bugs in order to avoid bug reopening later due to integration issues.
Researchers	Same as above	Current techniques on integration testing mainly focus on software bugs in general. Future research can propose novel integration testing techniques specifically for handling the reopened bugs to avoid extra effort in fixing such bugs.
	63% and 86% bugs have comments during and after bug reopening respectively. In 77% bugs the reason to reopen a bug is identified either during or after reopening the bug.	Researchers should leverage data from during/after bug reopening to determine challenges in reopened bug management.
	24% of the bugs are reopened to update the fields of the bug report.	Researchers should drop the sub-dataset containing bug reports reopened due to documentation (i.e., bookkeeping) issues.
	Prior studies on the prediction of reopened bugs are based on dataset with data leak issues.	Researchers should consider only those events that occur before a bug is reopened to predict reopened bugs.
	Only 34% studied projects give an acceptable performance on models trained to determine whether a bug will get reopened.	Current reopened bug prediction models are able to perform well when the majority of issues are reopened due to patch issues and integration issues as we observe in Section 3.2. However, when projects contain lots of reopened bugs due to non technical issues, the performance of reopened prediction models depreciates. Future research should consider better data filtering techniques when selecting projects to build reopened bug prediction models and there should be more research on how to build better reopened bug prediction models where reasons to reopen bugs are not primarily technical issues.
	The number of comments is the most important feature to predict reopened bugs.	Future studies on reopened bug prediction can leverage the number of developer comments as a factor to select reopened bugs for their study.

after committing the patch without waiting for the build to succeed, later the build fails leading to reopening the bug. To avoid build failures in the master branch, the developer can use a feature branch to test their patch before merging the patch to the master branch. For example, in a bug report[31] (ID: 1716) for the Apache Geode project, the developer tests the patch in feature branch, and later the bug is resolved. A better mechanism to control the

[31] https://issues.apache.org/jira/browse/GEODE-1716

quality of test/build processes in bug fixing is needed so that developers can avoid reopening bugs later on.

4.2 Implications for the researchers:

Researchers can propose novel integration testing techniques specifically for handling the reopened bugs to avoid extra effort in fixing such bugs. We observe in RQ2 (Section 3.2) that 13% of the manually studied bugs are reopened due to integration issues. Current techniques on integration testing mainly focus on software bugs in general (Herzig and Nagappan 2015; Rodríguez-Pérez et al. 2020). However, no prior research specifically focuses on integration testing techniques for reopened bugs. Therefore, we invite future research to conduct an experimental study in designing integration tests for reopened bugs so that the fixing process of reopened bugs can be improved.

Researchers should leverage data from during/after bug reopening to determine challenges in reopened bug management. We observe in RQ2 (Section 3.2) that, 63% of bugs have comments during reopening and 86% of bugs have comments after bug reopening. These during and after reopening comments which are not used in reopened bug prediction studies, can be used to understand developer discussions of reopened bugs in order to determine the cause of bug reopening and its remedy. For example, in a bug report[32] (ID: 5011) for the Apache Zepplin project, the bug is reopened and later the developer comments *"The build worked after I changed -Pspark-3.0 to -Pspark-3.0.0. At the end I learnt that it had ignored -Pspark-3.0.0 as an invalid option."* indicating the resolution of the reopened bug. In addition, we observe that 77% of our manually studied bugs have reason to reopen the bug during or after reopening the bug. A lot of this untapped data can be used to understand the challenges faced by developers in reopened bugs.

Researchers should drop the sub-dataset containing bug reports reopened due to documentation (i.e., bookkeeping) issues. We observe in RQ2 that one of the reason to reopen bugs is documentation issues. For example, in a bug report[33] (ID: 889) for the Apache HAWQ project, the developer reopened the bug to update the fix version. Some bugs are reopened in order to change a trivial bug report field, whereas other bugs are reopened in order to rework on the failed attempt to fix a bug. Bugs that are reopened due to documentation issues can be exempt from the reopened bug prediction model in order for the prediction model to predict more relevant bug reopening activities.

Researchers should consider only those events that occur before a bug is reopened to predict reopened bugs. We observe that in prior studies (Shihab et al. 2013; Xia et al. 2015), one out of three studied projects (i.e., Apache project) has bugs with the last status as *reopened* to determine the likelihood of reopening a bug. This is a data leak issue, and the bugs with a data leak issue should not be considered in the prediction model. We suggest the research community to carefully use data in prediction models, especially in the case when events occur sequentially.

Researchers should consider better data filtering techniques when selecting projects to build reopened bug prediction models and there should be more research on how to build better reopened bug prediction models where reasons to reopen bugs are not primarily technical issues. We observe in Section 3.1 that only 34% studied projects given an acceptable AUC on models trained to determine whether a bug will get reopened. Moreover, in

[32] https://issues.apache.org/jira/browse/ZEPPELIN-5011

[33] https://issues.apache.org/jira/browse/HAWQ-889

Section 3.2 that 98% of manually studied reopened bugs in projects with an acceptable AUC are due to patch and integration issues. However, in projects with poor AUC, the percentage of patch issues and integration issues drops by 10%. We invite future research to consider better data filtering techniques when selecting projects for predicting reopened bugs. For example, we recommend future research to select projects with more percentage of technical issues (Patch and integration issues). We also invite researchers to generate efficient models that are tuned to perform well on predicting reopened bugs on projects where bugs are reopened primarily due to non-technical issues (such as documentation issues, human issues).

Future studies on reopened bug prediction can leverage the number of developer comments as a factor to select reopened bugs for their study. We observe in Section 3.1 that the *number of comments* is the most important feature to predict reopened bugs. We suggest that researchers can leverage the number of developer comments and filter out reopened bugs with no/very less developer comments for their reopened bugs study.

5 Related Work

5.1 Bug Management

Bug triaging refers to the activities performed by the developers on un-assigned bugs, such as determining whether the bug needs attention, if so assigning the bug to a developer and adding a milestone for the bug (Bortis and Van Der Hoek 2013). Xia et al. (2016) proposed a MTM (multi-feature topic model) for bug triaging. Jalbert and Weimer (2008) proposed a system to automatically identify duplicate bugs. They observed that their system is able to reduce software development costs by removing 8% duplicate bugs. Tian et al. (2012) extended their work and improved the accuracy of duplicate bug detection. Somasundaram and Murphy (2012) recommended a list of components that are most relevant to a bug. Xia et al. (2014) found that most of the bugs set their priority and severity value as the default value. Prior studies also focussed on bug assignment tasks in the triaging process. Murphy and Cubranic (2004) proposed machine learning algorithms using text categorization to predict assignee of the bug. Other studies by Anvik et al. (2006), Xia et al. (2013), and Tian et al. (2016) proposed models to recommend the assignee for a bug. Our study also investigate bug management with respect to how bugs are reopened. We extend prior studies of bug reopening to conduct a large scale study with a modern machine learning model pipeline to understand the generalizability of reopened bug prediction models.

5.2 Predictive Models in Bug Management

Prior studies leveraged machine learning models to predict various attributes in a bug report. Shihab et al. (2010) conducted an initial study to predict reopened bugs on the Eclipse dataset using four dimensions and 22 features. Shihab et al. (2013) also extended their own work in another study to predict reopened bugs on three datasets (Apache, Eclipse, and OpenOffice). They built a Decision Tree model and performed the top node analysis to identify the most important features in predicting reopened bugs, and observed that *comment text* is the most important feature in predicting reopened bugs in the Eclipse project. Xia et al. (2013) investigated the performance of various classifiers to predict reopened bugs.

Among the 10 classifiers, they identified that Bagging and Decision Tree model gives the best performance. Xia et al. Xia et al. (2015) also showed that their results outperform the prior work. Xia et al. (2015) proposed the ReopenPredictor[34] tool to predict reopened bugs using the same dataset as used by Shihab et al. (2013). Xia et al. Xia et al. (2015) proposed a feature selection method for choosing the most substantial textual features (*comment text* and *description text*) for the imbalanced dataset from both reopened and non-reopened bugs. Xuan et al. (2012) modeled the developer prioritization of the bugs to predict the reopening of a bug. Xuan et al. (2014) discussed the benefits of data reduction techniques in reopened bug analysis. Caglayan et al. (2012) studied bug activity dataset in a large-scale enterprise software product. They analyzed four dimensions (i.e., issue-code relation, issue proximity network, issue reports, and developer activity) as possible factors that may cause a bug to be reopened. Zimmermann et al. (2012) conducted a study to find the root cause of reopened bugs and built a model to predict reopened bugs. Tu et al. (2018) defined data leak as information from the future used in predicting events making the prediction models misleadingly optimistic. They examined the prior literature in prediction tasks related to bugs and found that 11 out of 58 studies have data leak issues.

Prior studies on reopened bug prediction models focused only on a small dataset (i.e., three projects). Moreover, we identified that the Apache project used in the prior studies has a data leak issue. Instead, in our study, we use 47 projects to predict reopened bugs. We also use 7 classifiers in our study with latest techniques for generating reopened bug prediction model pipeline. Moreover, we also conduct a qualitative study to deeply understand the reasons why bugs get reopened.

5.3 Empirical Studies on Reopened Bugs

The activities of reopening bugs are an indication of issues that may project as more effort to resolve. Prior studies empirically mined bug reopening to gain a deeper understanding of reopened bugs. Shihab et al. (2013) conducted an investigation of the bug resolving time for reopened and non-reopened bugs in the Eclipse project. They observed that reopened bugs take at least twice as much time to resolve than non-reopened bugs (i.e., on average of 149 days for non-reopened bugs and 371 days for reopened bugs). Guo et al. (2010) conducted an empirical study on reopened bugs in Microsoft Windows and observed that if a bug is reopened too many times then probably it will not be fixed. Mi et al. (2018) discussed a new type of reopened bug that has no direct impact on the user experience or the normal operation of the system. They showed statistically that such reopened bugs are significantly different from other reopened bugs. Mi and Keung (2016) studied four open source projects, i.e., CDT, JDT, PDE, and Platform, to quantitatively analyze reopened bugs on their impacts, proportions, and evolution. They identified that 6% to 10% bugs are reopened and reopened bugs remain active between 1.6 and 2.1 times higher than non-reopened bugs. In our study, we observe more in-depth activities in the life cycle of reopened bugs. For example, we find that during bug reopening developers also change resolution (in 98% cases) and post comments (in 63% cases). Such activities can help identify the rationale for reopening a bug. We also observe that 12% of our studied bugs are reassigned during reopening. Future studies can investigate effects of reopening a bug on bug reassignment.

[34] https://github.com/xin-xia/reopenBug

6 Threats to Validity

6.1 External Validity

We use the JIRA bug tracking system for our study of reopened bugs. There are other popular bug tracking systems that can be studied and we encourage future work to investigate the characteristics of reopened bugs in such systems. Our selection criteria involve the top 50 projects in JIRA based on the committers count. The committers count criteria may not be the only metric to indicate the popularity of projects. For example, some committers may not be active, leading to projects with highly active but less number of committers. Future work can leverage other criteria that can be used to select projects. The selection of our studied dataset has an impact on the model performance and the feature importance analysis findings in our study.

6.2 Internal Validity

We study the reasons why bugs get reopened in RQ2 (Section 3.2). We select a sample of bugs for our qualitative analysis by examining 370 bug reports from 9,993 reopened bugs. Other bugs that are not part of our studied sample may contain reasons other than what we identified in our analysis. However, we minimize the sample bias by selecting a statistical representative sample with a 95% confidence level and a 5% confidence interval. In our study, we do not claim to find all the reasons to reopen a bug. To alleviate this threat, future research is encouraged to study a larger sample of reopened bugs.

6.3 Construct Validity

We considered 7 classifiers (i.e., Random Forest, Gradient Boosting, Logistic Regression, Adaboost, Multinomial Naive Bayes, Decision Tree, and KNN) to predict the reopened bugs. However, there are other classifiers that are not covered in determining the likelihood of reopening a bug. The selection of 7 classifiers for our study is motivated by the study conducted by Agrawal et al. (2019). Agrawal et al. used five classifiers (i.e., Random Forest, Logistic Regression, Multinomial Naive Bayes, Decision Tree, and KNN) for their study, we also consider two more classifiers (i.e., Gradient Boosting, and Adaboost) in our study as they are extensively used in prior prediction studies (Ghotra et al. 2015; Tantithamthavorn et al. 2016a, 2018).

We use the AUC value of 0.7 as a threshold to determine projects with an acceptable performance in determining the likelihood of reopening a bug. Changing the threshold value can change the percentage of projects with an acceptable performance, and impact the relative performance of different classifiers. However, we observe that many prior studies consider 0.7 as an AUC threshold to be acceptable (Jiarpakdee et al. 2020; Morasca and Lavazza 2020; Al Dallal and Morasca 2014; Turabieh et al. 2019).

7 Conclusion

A reopened bug can take considerably more time to fix, and developers can make a lot of effort in fixing reopened bugs. In this study, we revisit reopened bug prediction on a large dataset consisting of 47 projects tracked by JIRA and using an updated pipeline for reopened bug prediction. We share the findings of our paper: 1) Prior studies on reopened

bug prediction are based only on three projects (i.e., Apache, Eclipse, and OpenOffice), and out of these projects, the Apache project has a data leak issue. Hence, prior studies on predicting reopened bugs are effectively based only on two projects. Moreover, these studies used old techniques for constructing a reopened bug prediction model pipeline, indicating that predicting a reopened bug is not a solved problem. 2) Using our updated reopened bug prediction model pipeline, we observe that only 34% of our studied 47 projects have an acceptable performance (i.e., AUC $\geqslant 0.7$) in determining the likelihood of reopening a bug. 3) For the projects where the reopened bug prediction does not perform well, we propose researchers to create new features representing technical (i.e., specifically patch related) and documentation (i.e., bookkeeping) issues and leverage them to predict reopened bugs.

Based on the findings in our study, we suggest the developer community to follow practices involving carefully examining the proposed patch and testing the build success before resolving a bug. We find that 36% and 13% of the reopened bugs don't have explanations for reopening the bugs during and after reopening the bugs respectively. The bugs with bug reopening justifications are more likely due to technical issues, whereas the bugs without reopening justification can be due to trivial issues such as documentation. We invite future work to investigate reasons to reopen these bugs by conducting a deeper analysis such as developer surveys. We also encourage future research to explore this new venue of reopened bugs, e.g., by proposing new explanatory models to understand reopened bugs from leveraging the whole history of bug report and exploring insights into fixing reopened bugs more efficiently.

Acknowledgments We would like to thank the anonymous reviewers for their insightful comments. The findings and opinions in this paper belong solely to the authors and are not necessarily those of Huawei. Moreover, our results do not in any way reflect the quality of Huawei software products.

Declarations

Conflict of Interests The authors declare that they have no conflict of interest.

References

Agrawal A, Fu W, Chen D, Shen X, Menzies T (2019) How to" dodge" complex software analytics. IEEE Transactions on Software Engineering (TSE'19), pp 1–13

Agrawal A, Menzies T (2018) Is" better data" better than" better data miners"? In: 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE'18). IEEE, pp 1050–1061

Al Dallal J, Morasca S (2014) Predicting object-oriented class reuse-proneness using internal quality attributes. Empir Softw Eng (EMSE'14) 19(4):775–821

Altmann A, Toloşi L, Sander O, Lengauer T (2010) Permutation importance: a corrected feature importance measure. Bioinformatics 26(10):1340–1347

Anvik J, Hiew L, Murphy GC (2006) Who should fix this bug? In: Proceedings of the 28th international conference on Software engineering (ICSE'06), pp 361–370

Arellano AV (2019) Epidemiological disease surveillance using public media text mining. North Carolina State University

Artstein R, Poesio M (2008) Inter-coder agreement for computational linguistics. Comput Linguist 34(4):555–596

Beckler DT, Thumser ZC, Schofield JS, Marasco PD (2018) Reliability in evaluator-based tests: using simulation-constructed models to determine contextually relevant agreement thresholds. BMC Med Res Methodol 18(1):1–12

Biggers LR, Bocovich C, Capshaw R, Eddy BP, Etzkorn LH, Kraft NA (2014) Configuring latent dirichlet allocation based feature location. Empir Softw Eng (EMSE'14) 19(3):465–500

Bortis G, Van Der Hoek A (2013) Porchlight: A tag-based approach to bug triaging. In: 2013 35th International Conference on Software Engineering (ICSE). IEEE, pp 342–351
Boslaugh S (2012) Statistics in a nutshell: A desktop quick reference. O'Reilly Media, Inc.
Caglayan B, Misirli AT, Miranskyy A, Turhan B, Bener A (2012) Factors characterizing reopened issues: a case study. In: Proceedings of the 8th International Conference on Predictive Models in Software Engineering, pp 1–10
Cerda P, Varoquaux G, Kégl B (2018) Similarity encoding for learning with dirty categorical variables. Mach Learn 107(8-10):1477–1494
Chakraborty D, Elzarka H (2019) Advanced machine learning techniques for building performance simulation: a comparative analysis. J Build Perform Simul 12(2):193–207
Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP (2002) SMOTE: synthetic minority over-sampling technique. J Artif Intell Res 16:321–357
Chen D, Fu W, Krishna R, Menzies T (2018) Applications of psychological science for actionable analytics. In: Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp 456–467
Corazza A, Di Martino S, Maggio V, Scanniello G (2016) Weighing lexical information for software clustering in the context of architecture recovery. Empir Softw Eng (EMSE'16) 21(1):72–103
Corbin JM, Strauss A (1990) Grounded theory research: Procedures, canons, and evaluative criteria. Qualitative Sociol 13(1):3–21
da Costa DA, McIntosh S, Treude C, Kulesza U, Hassan AE (2018) The impact of rapid release cycles on the integration delay of fixed issues. Empir Softw Eng (EMSE'18) 23(2):835–904
Denny M, Spirling A (2017) Text preprocessing for unsupervised learning: Why it matters, when it misleads, and what to do about it. When It Misleads, and What to Do about It (September 27, 2017)
Fu W, Menzies T, Shen X (2016) Tuning for software analytics: Is it really necessary? Inf Softw Technol 76:135–146
Ghotra B, McIntosh S, Hassan AE (2015) Revisiting the impact of classification techniques on the performance of defect prediction models. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE'15), vol 1. IEEE, pp 789–800
Ghotra B, McIntosh S, Hassan AE (2017) A large-scale study of the impact of feature selection techniques on defect classification models. In: 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR'17). IEEE, pp 146–157
Guo PJ, Zimmermann T, Nagappan N, Murphy B (2010) Characterizing and predicting which bugs get fixed: an empirical study of microsoft windows. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE'10)-Volume 1, pp 495–504
Guo PJ, Zimmermann T, Nagappan N, Murphy B (2011) "not my bug!" and other reasons for software bug report reassignments. In: Proceedings of the ACM 2011 conference on Computer supported cooperative work, pp 395–404
Hardeniya N, Perkins J, Chopra D, Joshi N, Mathur I (2016) Natural language processing: python and nltk. Packt Publishing Ltd
Hayes AF, Krippendorff K (2007) Answering the call for a standard reliability measure for coding data. Commun Methods Measur 1(1):77–89
He H, Ma Y (2013) Imbalanced learning: foundations, algorithms, and applications. Wiley
Hébert A (2020) Estimation of road accident risk with machine learning. Ph.D. Thesis, Concordia University
Hemalatha I, Varma GPSaradhi, Govardhan A (2012) Preprocessing the informal text for efficient sentiment analysis. Int J Emerging Trends Technol Comput Sci (IJETTCS) 1(2):58–61
Herzig K, Nagappan N (2015) Empirically detecting false test alarms using association rules. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE'15), vol 2. IEEE, pp 39–48
Jalbert N, Weimer W (2008) Automated duplicate detection for bug tracking systems. In: 2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN). IEEE, pp 52–61
Jiarpakdee J, Tantithamthavorn C, Hassan AE (2019) The impact of correlated metrics on the interpretation of defect models. IEEE Transactions on Software Engineering (TSE'19)
Jiarpakdee J, Tantithamthavorn C, Treude C (2018) Autospearman: Automatically mitigating correlated software metrics for interpreting defect models. In: 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME'18). IEEE Computer Society, pp 92–103
Jiarpakdee J, Tantithamthavorn C, Treude C (2020) The impact of automated feature selection techniques on the interpretation of defect models. Empir Softw Eng (EMSE'20) 25(5):3590–3638
Kannan S, Gurusamy V (2014) Preprocessing techniques for text mining. Int J Comput Sci Commun Netw 5(1):7–16

Kaufman S, Rosset S, Perlich C, Stitelman O (2012) Leakage in data mining: Formulation, detection, and avoidance. ACM Trans Knowl Discov Data (TKDD'12) 6(4):1–21
Lee D, Rajbahadur GK, Lin D, Sayagh M, Bezemer C-P, Hassan AE (2020) An empirical study of the characteristics of popular minecraft mods. Empir Softw Eng (EMSE'20) 25(5):3396–3429
Lessmann S, Baesens B, Mues C, Pietsch S (2008) Benchmarking classification models for software defect prediction: A proposed framework and novel findings. IEEE Trans Softw Eng (TSE'08) 34(4):485–496
Li H, Shang W, Adams B, Sayagh M, Hassan AE (2020) A qualitative study of the benefits and costs of logging from developers' perspectives. IEEE Transactions on Software Engineering (TSE'20)
Lipton ZC (2018) The mythos of model interpretability. Queue 16(3):31–57
Malhotra R, Khanna M (2017) An empirical study for software change prediction using imbalanced data. Empir Softw Eng (EMSE'17) 22(6):2806–2851
McIntosh S, Kamei Y, Adams B, Hassan AE (2016) An empirical study of the impact of modern code review practices on software quality. Empir Softw Eng (EMSE'16) 21(5):2146–2189
McMillan C, Grechanik M, Poshyvanyk D, Fu C, Xie Q (2011) Exemplar: A source code search engine for finding highly relevant applications. IEEE Trans Softw Eng (TSE'11) 38(5):1069–1087
Méndez JR, Iglesias EL, Fdez-Riverola F, Díaz F, Corchado JM (2005) Tokenising, stemming and stopword removal on anti-spam filtering domain. In: Conference of the Spanish Association for Artificial Intelligence. Springer, pp 449–458
Meyer TA, Whateley B (2004) Spambayes: Effective open-source, bayesian based, email classification system. In: CEAS. Citeseer
Mi Q, Keung J (2016) An empirical analysis of reopened bugs based on open source projects. In: Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering, pp 1–10
Mi Q, Keung J, Huo Y, Mensah S (2018) Not all bug reopens are negative: A case study on eclipse bug reports. Inf Softw Technol 99:93–97
Morasca S, Lavazza L (2020) On the assessment of software defect prediction models via ROC curves. Empir Softw Eng (EMSE'20) 25(5):3977–4019
Murphy G, Cubranic D (2004) Automatic bug triage using text categorization. In: Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering. Citeseer, pp 1–6
Nyamawe AS, Liu H, Niu N, Umer Q, Niu Z (2020) Feature requests-based recommendation of software refactorings. Empir Softw Eng (EMSE'20) 25(5):4315–4347
Rajbahadur GK, Wang S, Ansaldi G, Kamei Y, Hassan AE (2021) The impact of feature importance methods on the interpretation of defect classifiers. IEEE Transactions on Software Engineering (TSE'21)
Rajbahadur GK, Wang S, Kamei Y, Hassan AE (2017) The impact of using regression models to build defect classifiers. In: 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR'17). IEEE, pp 135–145
Rajbahadur GK, Wang S, Kamei Y, Hassan AE (2019) Impact of discretization noise of the dependent variable on machine learning classifiers in software engineering. IEEE Trans Softw Eng (TSE'19):1–18
Rakha MS, Bezemer C-P, Hassan AE (2017) Revisiting the performance evaluation of automated approaches for the retrieval of duplicate issue reports. IEEE Trans Softw Eng (TSE'17) 44(12):1245–1268
Rodríguez-Pérez G, Robles G, Serebrenik A, Zaidman A, Germán DM, Gonzalez-Barahona JM (2020) How bugs are born: a model to identify how bugs are introduced in software components. Empir Softw Eng (EMSE'20) 25(2):1294–1340
Saha RK, Khurshid S, Perry DE (2015) Understanding the triaging and fixing processes of long lived bugs. Inf Softw Technol 65:114–128
Scikit-learn (2020) Tuning the hyper-parameters of an estimator. https://scikit-learn.org/stable/modules/grid_search.html#grid-search, [Online; accessed 08-June-2020]
Scoccia GL, Autili M (2020) Web frameworks for desktop apps: an exploratory study. In: Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM'20), pp 1–6
Shihab E, Ihara A, Kamei Y, Ibrahim WM, Ohira M, Adams B, Hassan AE, Matsumoto K (2010) Predicting re-opened bugs: A case study on the eclipse project. In: 2010 17th Working Conference on Reverse Engineering. IEEE, pp 249–258
Shihab E, Ihara A, Kamei Y, Ibrahim WM, Ohira M, Adams B, Hassan AE, Matsumoto Ki (2013) Studying re-opened bugs in open source software. Empir Softw Eng (EMSE'13) 18(5):1005–1042
Somasundaram K, Murphy GC (2012) Automatic categorization of bug reports using latent dirichlet allocation. In: Proceedings of the 5th India software engineering conference, pp 125–130
Song F, Liu S, Yang J (2005) A comparative study on text representation schemes in text categorization. Pattern analysis and applications 8(1-2):199–209

Srividhya V, Anitha R (2010) Evaluating preprocessing techniques in text categorization. Int J Comput Sci Appl 47(11):49–51
Stolberg S (2009) Enabling agile testing through continuous integration. In: 2009 agile conference. IEEE, pp 369–374
Tantithamthavorn C, Hassan AE, Matsumoto K (2018) The impact of class rebalancing techniques on the performance and interpretation of defect prediction models. IEEE Transactions on Software Engineering (TSE'18)
Tantithamthavorn C, McIntosh S, Hassan AE, Matsumoto K (2016) Automated parameter optimization of classification techniques for defect prediction models. In: Proceedings of the 38th International Conference on Software Engineering (ICSE'16), pp 321–332
Tantithamthavorn C, McIntosh S, Hassan AE, Matsumoto K (2016) An empirical comparison of model validation techniques for defect prediction models. IEEE Trans Softw Eng (TSE'16) 43(1):1–18
Tian Y, Sun C, Lo D (2012) Improved duplicate bug report identification. In: 2012 16th European Conference on Software Maintenance and Reengineering. IEEE, pp 385–390
Tian Y, Wijedasa D, Lo D, Le Goues C (2016) Learning to rank for bug report assignee recommendation. In: 2016 IEEE 24th International Conference on Program Comprehension (ICPC'16). IEEE, pp 1–10
Tu F, Zhu J, Zheng Q, Zhou M (2018) Be careful of when: an empirical study on time-related misuse of issue tracking data. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp 307–318
Turabieh H, Mafarja M, Li X (2019) Iterated feature selection algorithms with layered recurrent neural network for software fault prediction. Expert Syst Appl 122:27–42
Uysal AK, Gunal S (2014) The impact of preprocessing on text classification. Inf Process Manag 50(1):104–112
Vassallo C, Panichella S, Palomba F, Proksch S, Gall HC, Zaidman A (2020) How developers engage with static analysis tools in different contexts. Empir Softw Eng (EMSE'20) 25(2):1419–1457
Vieira R, da Silva A, Rocha L, Gomes JP (2019) From reports to bug-fix commits: A 10 years dataset of bug-fixing activity from 55 apache's open source projects. In: Proceedings of the Fifteenth International Conference on Predictive Models and Data Analytics in Software Engineering, pp 80–89
Webb JK, Keller KA, Welle K, Allender MC (2020) Evaluation of the inter-and intraindividual agreement of a pododermatitis scoring model in greater flamingos (phoenicopterus roseus). J Zoo Wildlife Med 51(2):379–384
Xia X, Lo D, Ding Y, Al-Kofahi JM, Nguyen TN, Wang X (2016) Improving automated bug triaging with specialized topic model. IEEE Trans Softw Eng (TSE'16) 43(3):272–297
Xia X, Lo D, Shihab E, Wang X, Zhou B (2015) Automatic, high accuracy prediction of reopened bugs. Autom Softw Eng (ASE'15) 22(1):75–109
Xia X, Lo D, Wang X, Yang X, Li S, Sun J (2013) A comparative study of supervised learning algorithms for re-opened bug prediction. In: 2013 17th European Conference on Software Maintenance and Reengineering. IEEE, pp 331–334
Xia X, Lo D, Wang X, Zhou B (2013) Accurate developer recommendation for bug resolution. In: 2013 20th Working Conference on Reverse Engineering (WCRE). IEEE, pp 72–81
Xia X, Lo D, Wen M, Shihab E, Zhou B (2014) An empirical study of bug report field reassignment. In: 2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE'14). IEEE, pp 174–183
Xuan J, Jiang H, Hu Y, Ren Z, Zou W, Luo Z, Wu X (2014) Towards effective bug triage with software data reduction techniques. IEEE Trans Knowl Data Eng 27(1):264–280
Xuan J, Jiang H, Ren Z, Zou W (2012) Developer prioritization in bug repositories. In: 2012 34th International Conference on Software Engineering (ICSE'12). IEEE, pp 25–35
Yadav A, Singh SK, Suri JS (2019) Ranking of software developers based on expertise score for bug triaging. Inf Softw Technol 112:1–17
Yatish S, Jiarpakdee J, Thongtanunam P, Tantithamthavorn C (2019) Mining software defects: should we consider affected releases? In: IEEE/ACM 41st International Conference on Software Engineering (ICSE'19). IEEE, pp 654–665
Zeng Y, Jiang K, Chen J (2019) Automatic seismic salt interpretation with deep convolutional neural networks. In: Proceedings of the 3rd International Conference on Information System and Data Mining, pp 16–20
Zhang H, Wang S, Chen T-H, Hassan AE (2020) Are comments on stack overflow well organized for easy retrieval by developers? ACM Trans Softw Eng Methodol (TOSEM'20) 29
Zhang H, Wang S, Chen T-H, Zou Y, Hassan AE (2019) An empirical study of obsolete answers on Stack Overflow. IEEE Transactions on Software Engineering (TSE'19)

Zheng A, Casari A (2018) Feature engineering for machine learning: principles and techniques for data scientists. O'Reilly Media, Inc.

Zimmermann T, Nagappan N, Guo PJ, Murphy B (2012) Characterizing and predicting which bugs get reopened. In: 2012 34th International Conference on Software Engineering (ICSE'12). IEEE, pp 1074–1083