Factors Characterizing Reopened Issues: A Case Study

Bora Caglayan[1], Ayse Tosun Misirli[2], Andriy Miranskyy[3], Burak Turhan[4], Ayse Bener[5]

Bogazici University[1,2], IBM Canada Ltd.[3], University of Oulu[4], Ryerson University[5]
Department of Computer Engineering, Istanbul, Turkey[1,2]
IBM Toronto Software Laboratory, Toronto, ON Canada[3]
Department of Information Processing Science, Oulu, Finland[4]
Ted Rogers School of Information Technology Management, Toronto, ON Canada[5]
{bora.caglayan[1], ayse.tosun[2]}@boun.edu.tr
andriy@ca.ibm.com [3], burak.turhan@oulu.fi [4], ayse.bener@ryerson.ca[5]

ABSTRACT

Background: Reopened issues may cause problems in managing software maintenance effort. In order to take actions that will reduce the likelihood of issue reopening the possible causes of bug reopens should be analysed.
Aims: In this paper, we investigate potential factors that may cause issue reopening.
Method: We have extracted issue activity data from a large release of an enterprise software product. We consider four dimensions, namely *developer activity*, *issue proximity network*, *static code metrics* of the source code changed to fix an issue, *issue reports and fixes* as possible factors that may cause issue reopening. We have done exploratory analysis on data. We build logistic regression models on data in order to identify key factors leading issue reopening. We have also conducted a survey regarding these factors with the QA Team of the product and interpreted the results.
Results: Our results indicate that centrality in the issue proximity network and developer activity are important factors in issue reopening. We have also interpreted our results with the QA Team to point out potential implications for practitioners.
Conclusions: Quantitative findings of our study suggest that issue complexity and developers workload play an important role in triggering issue reopening.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—process metrics, complexity measures, performance measures

General Terms

Measurement, Experimentation

PROMISE '12, September 21–22, 2012, Lund, Sweden

Keywords

software maintenance, issue management, issue repository, issue reopening.

1. INTRODUCTION

Issue management is a key activity within software projects especially during the maintenance phase. A steady influx of new issue reports are filtered, prioritized, assigned and handled by the maintainers daily for popular software. Managing the issue handling process effectively is an important element for the long term stability of the software product.

In an idealized issue management scenario, the issue report and the records of issue activity are moved to the archives indefinitely after an issue is closed. Reopened issues are a group of issues that are exceptions to this idealized scenario. These issues go through the issue handling procedures at least one more time after they are archived. Understanding reopened issues is of significant interest to practitioners, since these issues may represent a miscommunication between issue assigner and assignee. Furthermore, reopened issues may cause waste of time and effort if they are frequent in issue repositories.

We asked the the quality assurance (QA) team of a large scale company about the possible reasons of reopening and the benefits of investigating these reasons. The opinions of the QA team are as follows:

- Reopened issues may have different explanations, but the major one is a back and forth discussion between issue originator and owner on "yes, it's bug /no, it works as designed". Most of the time, people spent arguing on the issue and it is assigned to "opened" multiple times due to this discussion.
- Reopened issues consist of less than 10% of all issues, and hence it is not a major pain point. But, they also cause waste of resources. Identifying the main reasons for these issues gives us a chance for process improvement. Reducing this ratio down to 5% would also be very beneficial.

We consider an issue as reopened if it has changed from a termination state such as closed, cancelled or postponed to an active state such as assigned or work-in progress. Identifying the factors that lead to issue reopens is crucial in such a situation in order to take the necessary actions.

In this paper our research question is identifying the possible factors that may lead to issues getting reopened:

- RQ: Which factors lead to issues getting reopened?

In order to answer our research question, we analysed a large scale enterprise software and its issue and code repository. We modelled four dimensions, namely 1)issue-code relation, 2)issue proximity network, 3)issue reports and 4) developer activity, in order to check their individual effects on issues getting reopened.

In the analysis, we used logistic regression to fit a predictive model on the issue data. As a first step, we conducted univariate regression by using each of the factors from the four dimensions namely developer activity, issue-code relation, issue proximity network and the issue report. After that, we did an exhaustive search for the best factor combination by optimizing the AUC (area under the receiver operating characteristic curve) for all the possible factor combinations. Finally, we showed the model with the highest success rate among all possible model combinations.

Previously two independent research groups investigated the factors that may cause issue reopens for Microsoft Windows [25] and Eclipse projects [19]. However, to the best of our knowledge, factors used in our study related to issue-code relation and issue proximity network have not been considered previously. The main contribution of this paper is analysis of factors that lead to issues getting reopened in a large-scale software developed in multiple locations. Since some of the measures used in the two previous studies are not extractable for our dataset, our aim is complementing the findings of previous researchers rather than testing their findings on our new dataset.

The rest of the paper is structured as follows: In Related Work section, we discuss the relevant work on understanding the reasons for issue reopening. In Methodology, we present the dataset, data extraction process and metrics and logistic regression models. In Results section, we show and interpret the outcomes of the logistic regression model. Finally a discussion of threats to the validity of the results and conclusions with possible future research topics are presented.

2. METHODOLOGY

In this paper, we perform quantitative analysis on issue activity database, whose attributes are described in Section 2.1, extracted factors that may have significant influences on reopened issues (Section 2.2), and built a statistical model to interpret them (Section 2.3).

2.1 Dataset

We have used issue activity database of a large-scale enterprise software product which has a long development history with a 20 years old code base. The company uses IBM Rational ClearQuest with customised defect forms as the issue management system. In this database, each issue record includes, but not limited to, the following features:

- **Originator:** The person who opens an issue. Often, testers (or support personnel) are the originators.
- **Owner:** The person who is assigned to an issue. The owner is often the developer, who fixes an issue especially when issue is classified as a *defect*.
- **State:** There are 11 distinct states in the database of the company: Opened, Assigned, Working, Delivered, Returned, Integrated, Validated, Rejected, Closed, Postponed, Cancelled.
- **Phase Found:** It indicates the phase in the development life cycle an issue is reported. A list of phases are summarized in Appendix with their occurrence rates in the issue database.
- **Symptom:** This is a sign of problem ("crash/outage") experienced by a customer. Some of the common symptoms are Build/Test Failed, Core Dump, Program Defect, Incorrect I/O. We have found that symptoms are significantly correlated with *phase found*.

A typical life cycle of an issue in our case study can be seen in Figure 1. Bold arrows show a typical life cycle, while the dashed arrows numbered with (1), (3), (4) and (5) indicate a *reopening*. It is often the case when an issue is assigned right from the start and the owner starts working on it immediately (arrow number (2)). The final status of an issue is stored in *State* field, but changes of the state are stored in two other fields (*old state* and *new state*) with the person id (generally *owner* or *originator*) who makes this modification.

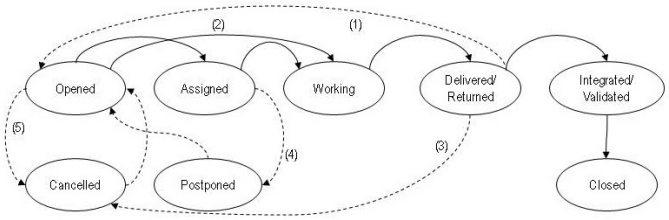

Figure 1: Life cycle of an issue in our case study

In our case study, issue activity database contains 3645 unique issue reports with the earliest record opened on January, 2005. We filtered only closed issues from this database and obtained 2287 issues in total. Of these issues, 219 (approximately 9%) are classified as *reopened*. This dataset is further filtered as we incorporate factors from the code base.

2.2 Factors Affecting Issue Reopening

We have previously extracted code, network and churn metrics from the code base of the same product 16 months prior to release date at the method level. Our data extraction methodology and prediction models built with these metric sets can be read from [8] and [18]. In this paper, we identify four dimensions about 1) developers who fix these issues, 2) size and complexity of methods edited during issue fixes, 3) the relationship between issues and 4) other factors about issue reports and fix activities. For each of these factors, we define hypotheses. We also define abbreviations for all factors to ease their usage on tables and figures.

Of these 2287 closed issues in the database, 1318 issue records are matched with developer activities in which there are 88 (7%) reopened issues. Furthermore, when code and network metrics are extracted, these issues are mapped with source code at method level. After issue-code mapping, final dataset includes 1046 issues, of which 62 (6%) are reopened.

2.2.1 Developer Activity

Previously, Guo et al. [12] and Zimmermann et al. [25] defined bug opener's reputation as the ratio of *"total number of previous bugs opened and gotten fixed"* over *"total number of bugs he/she opened"*. In a case study on Firefox performance and security bugs, authors analysed "who fixes these bugs" by measuring the expertise of developers in terms of *number of previously fixed bugs by the developer* and *experience in days*, i.e. number of days from the first fix by the given developer to the latest bug's fix date [23]. Similar to these approaches, we extracted 6 different metrics representing the development activities of issue owners, i.e., developers who owned and fixed issues.

We analyse developers' defect fixing activity to verify the following hypotheses:

H_1: Issues owned by developers, who fix few issues, are more likely to be reopened.

H_2: Issues owned by developers, who haven't fixed an issue for a long time, are more likely to be reopened.

H_3: Issues owned by developers, who edit large number of methods to fix an issue, are more likely to be reopened.

 # Fixed Issues (dev_fix_count): To test H_1, we computed the number of previously fixed issues of a developer who owns the current issue, up to the current issue's opening date. For example, suppose john.black@XXX.com owns issue ID #120 which was opened on May 2009. Then, we calculated the number of previously fixed issues by john.black since May 2009. Therefore, even though two issues are fixed by the same developer, this metric's value may change for these issues if their opening dates are different from each other.

Rank in terms of # Fixed Issues (dev_rank_fix_count): This metric is calculated using *Fixed Issue Count* and ranks of developers who owned and fixed issues. For each issue's opening date, # fixed issues and ranks by developers are re-calculated. For example, suppose john.black@ XXX.com fixed issue ID #120 which was opened on May, 5th 2009 and closed on June, 6th 2009, and he also fixed issue ID #133 which was opened on July, 5th 2009 and closed on October, 6th 2009. Considering this case, we computed previously fixed issues for john.black twice, both for May 2009 (k) and July 2009 (k+1 by adding issue #120). Ranks of this developer is also calculated based on other developers' issue fix performance on May 2009 and July 2009. So, john.black can be at the first rank on May 2009 with k fixed issues, but at the third rank on July 2009 with *(k+1)* fixed issues.

Duration between the First and Last Fix (dev_first_last_fix): This metric computes the number of days from the first fix of the developer to the current issue's fix date. To test H_2, we computed this metric for all developers associated with issues in our database. The reason for choosing this metric is as follows: Reopened issues are critical in the sense that they require thorough knowledge of the source code and developers who would fix reopened issues also require active development history to avoid forgetting possible bottlenecks in the source code. If the duration between the first and last fix of a developer is long, then developer may spend harder time during understanding the main reasons for the issue in an updated software system and this may cause issues to be re-opened.

Total # of Edits (dev_total_edits): To test H_3, we extracted total # of edits a developer has done on a method (i.e. function) from development commit logs and associated them with issues. The more edits are done on software methods by a developer, the more likely the developer is well informed about the software system and the more likely he/she is an active developer.

 # Unique Methods Edited (dev_methods_edited): To test H_3, we have also considered unique number of methods edited for fixing an issue. Total # of edits is not enough to evaluate whether developers of reopened issues have a strong code ownership. If a developer edits majority of methods in software, it may indicate his/her ownership on the code. Having strong ownership may also avoid issue reopenings since it also suggests that the issue owner has an extensive knowledge on the source code as well as potential problems.

Rank in terms of # Methods Edited (dev_rank_methods_count): This metric is calculated based on *unique methods edited* and ranks of developers who owned and fixed an issue are computed. For each issue, its owner's (developer) rank in terms of number of methods he/she edited so far is computed and added as a new metric. Computation of these ranks is similar to *rank in terms of # fixed issues*. This metric also completes the general definition of *code ownership* by adding both the number of methods edited by a developer as well as what percentage of edits are done by this developer among all developers (i.e., developer's rank).

2.2.2 Issue-Code Relation

Shihab et al. [19] considered the fact that re-opened bugs may be harder to fix than others due to the fact that they require many files or more complex files to be changed. We have also considered the complexity of reopened issues in terms of software methods changed during their fixes and define two hypotheses:

H_4: Issues related with many methods are more likely to be reopened.

H_5: Issues related with larger (in terms of lines of code) and more complex methods are more likely to be reopened.

 # Methods Changed (methods_changed): This metric is calculated by counting the number of methods changed for fixing an issue by mining commit messages from version control systems and matching each commit with an issue. It is then used to test H_3.

LOC: Number of methods changed during a fix may not be enough to represent the complexity of an issue. For example, an issue may require changes on 3 methods, but each method may be greater than 100 lines of code (LOC) and hence its fix may be harder than other fixes. Therefore, we have also extracted a size indicator (in terms of lines of code) for methods changed for fixing an issue. If there are more than one method changed during a fix, we aggregated their lines of codes by taking maximum and sum values over all methods.

Cyclomatic complexity (CC): As an extension to LOC measure, we have used McCabe's cyclomatic complexity of a method changed for fixing an issue. If there are more than one method changed, we aggregated their cyclomatic complexity values by taking maximum and sum values over all methods. This metric as well as *LOC changed* are used to test H_4.

2.2.3 Issue Proximity Network

Issue proximity network models the *relation* between issues. It measures the *distance between issues* in terms of

the *number of common methods changed during their fixes*. If an issue is connected with many other issues in terms of the number of common methods changed during a fix, it may increase the probability of this issue being reopened. The reason for this can be explained as follows: Reopened issues may have close connections with many other issues and therefore they reside at the center of this proximity network. However, being in the core part also indicates that reopened issues may affect many methods in the source code, which increases the risk of failures afterwards. We define our hypothesis for measuring this dimension as follows:

H_6: Issues linked with many other issues are more likely to be reopened.

Metrics are extracted from issue proximity network, all of which were used in previous studies [18] to measure caller-callee relations between software modules and their effects on defect proneness. In this paper, we used four network metrics to quantify complexity of issues and how complexity (in terms of methods changed) is related to issue reopening.

Degree: This metric is computed by counting the number of direct relations (edges) an issue has. Having higher degree means that an issue is connected to many other issues, such as a hub in traffic networks.

Degree Centrality: In our previous studies, we have extracted both in-degree and out-degree centrality metrics [8, 18]. However, in this paper, the proximity network is *undirected* with weights of edges are set as the number of methods shared by two issues. Therefore, this metric is calculated by "degree" of an issue over all issues ($degree/N$ where N is number of issues).

Betweenness Centrality: This metric is calculated by counting the number of shortest paths that contain the issue X over all shortest paths between all issue pairs, *i,j*. It evaluates the location of an issue, since being in a popular location may be very critical due to the fact that an issue has association with many issues as well as it affects many methods in the source code.

Pagerank: This metric measures the relative importance of an issue. It also evaluates the centrality in issues by considering the fact that the effect of being related with a central issue should be more important than being related with a decentralized issue.

2.2.4 From Issue Reports

From issue reports, we have extracted 2 categorical metrics, namely *Symptom* and *Phase found*. Our objective is to observe whether reopened issues have unique symptoms or they are more likely to be reported during a specific phase.

Same Location (same_loc): We have also extracted geographical locations of the owner and originator of issues based on their email addresses' domain and defined a boolean metric, *Same location* to observe the effect of communication across different locations on issue re-openings. Our hypothesis to test this relation is as follows:

H_7: Issues whose owner and originator are from different locations are more likely to be reopened.

In a study done by Herbsleb and Mockus [14], it was found that issues reported (and fixed) in distributed teams have a higher resolution time than issues reported and fixed in the same location. Zimmermann et al. also investigated the effects of location differences between assigners and assignees on reopened bugs and found that bugs initially assigned across teams/ buildings or countries are more likely

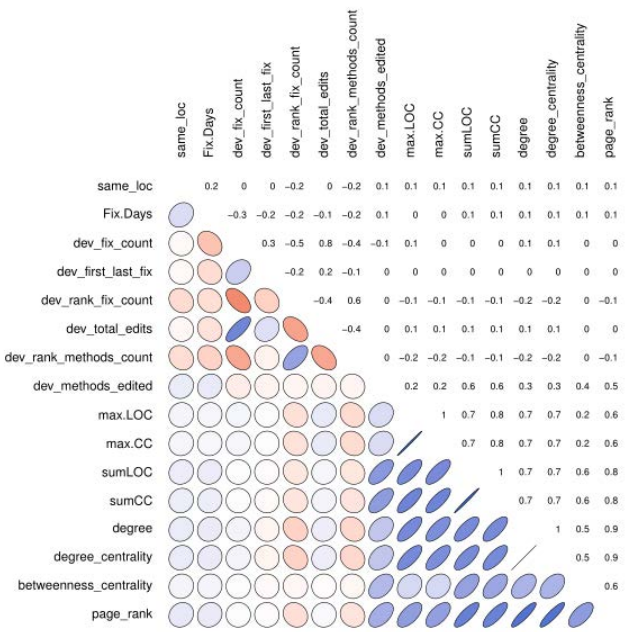

Figure 2: Correlations of the measures. The shape of the ellipse represents the correlation among two variable. In the correlation visualisation, bolder colors indicate higher correlations. If the shape of an ellipse bends towards right, it indicates positive correlation, whereas negative correlation if its shape bends towards left.

to be reopened [25]. Thus, we have defined *Same location* and assigned 1 if both originator and owner of an issue were located in the same country, and 0 in the opposite case. Based on the data, issues were reported from 12 distinct geographical locations. Only 20% of issues had their owners and originators being in different locations.

Fix Days: Reopened issues may have a long life cycle from their opened to closed dates since they were assigned to same states (opened/ assigned) more than once. We have defined a new metric, namely *Fix days*, to measure the number of days between an issue's opened and closed date. Hypothesis to test the effect of this metric is defined as follows:

H_8: Issues which take long time (in days) to fix are more likely to be reopened.

2.3 Analysis of The Factors

Basic descriptive statistics of the factors can be found in Table 1. Median values of factors for reopened and not-reopened issues are significantly different (Mann-Whitney U Test $P < 0.05$) in 10 out of 19 cases. This shows that distributions of reopened issues are shifted towards less/ more activity in each of 10 factors separately. For instance, reopened issues cause significantly more edits on the source code (8^{th} factor in Table 1) compared to other issues.

Descriptive statistics of factors related to issue-code relation dimension are particularly interesting. In the extreme cases, an issue fix can change up to 460 methods or files with up to 92 kLOC. We believe that such a pattern highlights relative complexities of addressed issues or architectural complexities of the software. A fix in a complex software will

likely involve changes at a lot of interdependent modules. Project issue fix count distribution is similar to the Pareto-Law trends observed in the developer activity distribution in open source projects [15].

Spearman rank correlation coefficients among the factors we considered are visualised in Figure 2. In the correlation visualisation, bolder colors indicate higher correlations. If the shape of an ellipse bends towards right, it indicates positive correlation, whereas negative correlation if its shape bends towards left. From the figure, it can be observed there are relatively higher correlations among the factors within the same dimension, while there are relatively lower correlations between the factors from different dimensions. The high correlations are especially apparent among the factors from the issue-code relation (LOC, CC) and issue proximity (degree, betwenness, pagerank) dimensions.

2.4 Logistic Regression Models

Univariate Logistic Regression

Logistic regression is the standard way to model binary outcomes $y_i = 0, 1$ [1, 10] and therefore it is suitable for our problem. Logistic regression has been frequently used in classification problems such as defect prediction in the software engineering literature previously [22, 24]. The basic probability model formula of logistic regression can be stated as follows:

$$Pr(y_i = 1) = logit^{-1}(X_i\beta) \quad (1)$$

$Pr(y_i = 1)$ is the probability of outcome $y_i = 1$. X_i is a vector of independent parameters for the instance and *beta* is the vector of regression coefficients.

One advantage of logistic regression in binary classification when compared to methods like Naive Bayes is that its regression coefficients and other parameters (odds ratio) are easily interpretable and highly explicative. Assuming the logistic regression model is true, one can check the significance of various regression coefficients of different input variables to understand their explanatory power. In addition relation of odds, $(y_i = 1/y_i = 0)$ and individual factors can be analysed by the following formula:

$$Pr(y_i = 1)/Pr(y_i = 0) = e^{X_i\beta} \quad (2)$$

From this formula, the effect of various factors on the probability of a certain outcome y_i can be understood by analysing the effects of their changes.

Exhaustive Search on All Factor Combinations

In order to test the performance of logistic regression with all possible factors (N), 2^N models should be considered. For large data, various approaches are used to reduce the number of considered combinations. However, we were able to test all possible combinations of models (524,288 in total) in a couple of hours, since our dataset was relatively small.

In order to find the "best" model, a performance criterion should be optimized. In the literature, a likelihood based information criteria such as AIC(Akaike Information criterion) or BIC (Bayesian Information criterion) is often used to maximize the likelihood function while penalizing overfitting [17]. Instead of a maximum likelihood based performance measure, we use Area Under the receiver operational characteristic Curve (AUC) as the performance measure. We believe that AUC represents the predictive performance of the model more clearly than a likelihood based measure. AUC is commonly used to compare the performance of various classification models [16].

Multivariate Logistic Regression

As a second model we built a logistic regression model using a set of factors with the best predictive power. We observed the predictive power of this model by drawing its ROC curve and reported the factors included to the model.

3. RESULTS OF LOGISTIC REGRESSION MODELS

3.1 Univariate Statistical Model

In order to check the importance of factors individually, we have checked the significance of univariate logistic regression models [7]. In Table 2, factors with significant correlations are presented. Five out of 19 factors were found to have significant correlation coefficients. One of these factors are related to issue report (same location), three are related to the developer activity (dev*first_last_fix, dev_total_edits, dev_methods_edited) and one is related to issue proximity network (betweenness centrality). We have interpreted our hypotheses by building a univariate regression model to predict reopened issues with 19 factors respectively. Regression coefficients and their significance regions reported in Table 2 are used to validate if a factor is significant for predicting reopened issues. In summary, hypotheses related with developer activity in terms of frequent issue fixes and methods edited (H_2, H_3), issue proximity network (H_6) and geographical locations of issue owner and originator (H_7) are validated and these findings are summarized below in bold and italic. Other hypotheses could not be validated with univariate analysis, but we have also checked their significance using multivariate analysis of factors in later sections.

Developer activity: We have found that reopened issues have a significantly negative relation with developers who have not fixed an issue for a long time (coefficient: -2.6947136, $p \ll 0.05$). Furthermore, reopened issues have a positive relation with developers who edit relatively more methods (coefficients: 0.0012903, 0.0019286, $p < 0.05$). These results validate our second and third hypotheses (H_2, H_3). But, we could not validate H_1, which defines a relationship between developer activity in terms of *previously fixed issues* and issue *reopening*.

Reopened issues are fixed by developers who actively fix issues and edit many methods.

Issue proximity network: Regarding issue relations in terms of shared methods, we have validated H_6 with the highest coefficient (35.8880) on betweenness centrality.

Reopened issues are linked with other issues in terms of methods edited during issue fixes.

Issue report: Issues whose owner and originator located in the same location have a significantly negative impact on issue reopening (coefficient: -1.1320, $p \ll 0.05$). This also validates H_7, since being in different geographical locations may lengthen the communication process and cause issue reopening. However, we could not validate the relationship between *fix days* and reopened issues (H_8).

Reopened issues are often reported and fixed by people from different geographical locations.

Table 1: Descriptive Statistics of The Considered Factors. The first values in the cells are for all the issues. Values in parantheses are: α for not reopened issues, β for reopened issues.♣: Factor has significantly different medians for reopened and not reopened issues

	Factor	Max	75 %	Median	25%	Minimum	Mean
Issue Report	Symptom	-	-	-	-	-	-
	Phase Found	-	-	-	-	-	-
	Same Location	-	-	-	-	-	-
	Fix Days	3745(3745$^{\alpha}$ 981$^{\beta}$)	240(241$^{\alpha}$ 217$^{\beta}$)	119(121$^{\alpha}$ 81$^{\beta}$)	39(40$^{\alpha}$ 30$^{\beta}$)	0(0$^{\alpha}$, 4$^{\beta}$)	169(168$^{\alpha}$, 176$^{\beta}$)
Developer Activity	# Fixed Issues ♣	79(79$^{\alpha}$ 58$^{\beta}$)	22(21.25$^{\alpha}$ 27.5$^{\beta}$)	11 (11$^{\alpha}$ 13$^{\beta}$)	4(4$^{\alpha}$ 4$^{\beta}$)	1(1$^{\alpha}$, 1$^{\beta}$)	15.5 (15.38$^{\alpha}$ 18.29$^{\beta}$)
	Rank in # Fixed Issues ♣	125(119$^{\alpha}$ 125$^{\beta}$)	31(31$^{\alpha}$ 33.25$^{\beta}$)	14(14$^{\alpha}$ 14.5$^{\beta}$)	5(6$^{\alpha}$ 4$^{\beta}$)	1(1$^{\alpha}$, 1$^{\beta}$)	21.49(21.31$^{\alpha}$, 24.45$^{\beta}$)
	Duration Between First and Last Fix in Days ♣	3786(3786$^{\alpha}$ 833$^{\beta}$)	478(475.25$^{\alpha}$ 559$^{\beta}$)	282.5(279$^{\alpha}$ 373$^{\beta}$)	124(124$^{\alpha}$ 130.75$^{\beta}$)	0(0$^{\alpha}$, 0$^{\beta}$)	388(391$^{\alpha}$, 345$^{\beta}$)
	Total # Edits ♣	1347(1347$^{\alpha}$ 1167$^{\beta}$)	180(177.25$^{\alpha}$ 241$^{\beta}$)	80(80$^{\alpha}$ 115$^{\beta}$)	24(24$^{\alpha}$ 19.5$^{\beta}$)	0(0$^{\alpha}$, 0$^{\beta}$)	149(144$^{\alpha}$, 234$^{\beta}$)
	# Unique Methods Edited ♣	864(864$^{\alpha}$ 794$^{\beta}$)	113(110$^{\alpha}$ 168.75$^{\beta}$)	56(55$^{\alpha}$ 68$^{\beta}$)	20(20$^{\alpha}$ 17.5$^{\beta}$)	0(0$^{\alpha}$, 0$^{\beta}$)	99(95$^{\alpha}$, 161$^{\beta}$)
	Rank in # Methods Edited	129(129$^{\alpha}$ 115$^{\beta}$)	34(34$^{\alpha}$ 28.5$^{\beta}$)	14(15$^{\alpha}$ 10.5$^{\beta}$)	4(4.75$^{\alpha}$ 1$^{\beta}$)	0(0$^{\alpha}$, 0$^{\beta}$)	21(22$^{\alpha}$, 20$^{\beta}$)
Issue-Code Relation	Max. CC ♣	2103(2087$^{\alpha}$ 2103$^{\beta}$)	275(309.3$^{\alpha}$ 131.75$^{\beta}$)	91(92$^{\alpha}$ 75.5$^{\beta}$)	37(37$^{\alpha}$ 35.25$^{\beta}$)	1(1$^{\alpha}$, 4$^{\beta}$)	303.3(307.2$^{\alpha}$, 240.9$^{\beta}$)
	Sum CC ♣	2103(2087$^{\alpha}$ 2103$^{\beta}$)	275(309.3$^{\alpha}$ 131.75$^{\beta}$)	91(92$^{\alpha}$ 75.5$^{\beta}$)	37(37$^{\alpha}$ 35.25$^{\beta}$)	1(1$^{\alpha}$, 4$^{\beta}$)	303.3(307.2$^{\alpha}$, 240.9$^{\beta}$)
	Max. LOC ♣	22644(22644$^{\alpha}$ 22567$^{\beta}$)	3077(3643$^{\alpha}$ 1609$^{\beta}$)	1152(1170$^{\alpha}$ 863.5$^{\beta}$)	468(466$^{\alpha}$ 473$^{\beta}$)	28(28$^{\alpha}$, 146$^{\beta}$)	3458(3508$^{\alpha}$, 2680$^{\beta}$)
	Sum LOC ♣	92931(92931$^{\alpha}$ 67509$^{\beta}$)	7543(7730$^{\alpha}$ 4588$^{\beta}$)	2014(2023$^{\alpha}$ 1586$^{\beta}$)	646(645$^{\alpha}$ 721$^{\beta}$)	28(28$^{\alpha}$, 146$^{\beta}$)	6076(6103$^{\alpha}$, 5648$^{\beta}$)
	# Methods Changed	460(460$^{\alpha}$ 86$^{\beta}$)	9(9$^{\alpha}$ 10$^{\beta}$)	3(3$^{\alpha}$ 5$^{\beta}$)	1(1$^{\alpha}$ 1.25$^{\beta}$)	1(1$^{\alpha}$, 1$^{\beta}$)	11.26(11.25$^{\alpha}$, 11.42$^{\beta}$)
Issue Proximity Network	Degree ♣	173(115$^{\alpha}$ 173$^{\beta}$)	31(31$^{\alpha}$ 26$^{\beta}$)	10(10$^{\alpha}$ 11$^{\beta}$)	3(3$^{\alpha}$ 4$^{\beta}$)	0(0$^{\alpha}$, 0$^{\beta}$)	20.32(20.02$^{\alpha}$, 25.13$^{\beta}$)
	Betweenness Centrality	0.22(0.08$^{\alpha}$ 0.22$^{\beta}$)	0.0009(0.0009$^{\alpha}$ 0.002$^{\beta}$)	0.0002(0.0002$^{\alpha}$ 0.0006$^{\beta}$)	$1.28e^{-07}$(0$^{\alpha}$ $1.23e^{-05}$ $^{\beta}$)	0(0$^{\alpha}$, 0$^{\beta}$)	0.002(0.002$^{\alpha}$, 0.008$^{\beta}$)
	Pagerank	0.007(0.006$^{\alpha}$ 0.007$^{\beta}$)	0.001(0.001$^{\alpha}$ 0.002$^{\beta}$)	0.0008(0.0008$^{\alpha}$ 0.0008$^{\beta}$)	0.0004(0.0004$^{\alpha}$ 0.0005$^{\beta}$)	0(0$^{\alpha}$, 0$^{\beta}$)	0.0001(0.0001$^{\alpha}$, 0.001$^{\beta}$)
	Degree Centrality ♣	0.18(0.12$^{\alpha}$ 0.18$^{\beta}$)	0.03(0.03$^{\alpha}$ 0.03$^{\beta}$)	0.01(0.01$^{\alpha}$ 0.01$^{\beta}$)	0.003(0.003$^{\alpha}$ 0.004$^{\beta}$)	0(0$^{\alpha}$, 0$^{\beta}$)	0.02(0.02$^{\alpha}$, 0.02$^{\beta}$)

Table 2: Coefficients for Univariate Regression

Dimension	Factor	Coefficient	Standard Deviation	Z Value	$Pr(>\|z\|)$	Significance
Issue Report	Symptom	-	-	-	2	
	Phase Found	-	-	-	2	
	Same Location	-1.13200	0.27000	-4.160	3.14e-05	★★★
	Fix Days	0.00018	0.00061	0.296	0.7700	
Developer Activity	Dev_fix_count	0.01100	0.00770	1.460	0.1500	
	Dev_rank_fix_count	0.00640	0.00560	1.130	0.2600	
	Dev_first_last_fix	-2.69000	0.17000	-16.222	<2e-16	★★★
	Dev_total_edits	0.00130	0.00043	3.030	0.0024	★★
	Dev_methods_edited	0.00190	0.00060	3.200	0.0014	★★
	Dev_rank_methods_count	-0.00250	0.00600	-0.420	0.6700	
Issue-Code Relation	Max. CC	-0.00031	0.00031	-0.995	0.3200	
	Sum CC	-7.08e-05	1.69e-04	-0.420	0.6800	
	Max. LOC	-3.30e-05	2.89e-05	-1.140	0.2500	
	Sum LOC	-4.96e-06	1.42e-05	-0.350	0.7300	
	Methods_changed	0.00022	0.00460	0.047	0.9600	
Issue Proximity Network	Degree centrality	7.45000	4.63000	1.610	0.1100	
	Degree	0.00760	0.00469	1.609	0.1080	
	Betweenness centrality	35.89000	12.01700	2.990	0.0028	★★
	Pagerank	21.16000	11.80000	2.130	0.2200	

Issue-code relations: Univariate analyses do not show a significant relation between code metrics and reopened issues. Hence, we could not validate H_4, H_5, but observed predictive power of code metrics in multivariate regression models.

3.2 The Best Factor Combinations

We ranked all possible combinations of the factors (2^{19} possible models) to predict reopened issues based on our performance measure, AUC. We counted the occurrence of each factor in the best 100 models. Best 100 models had a AUC difference of 0.05 between the best performing and the worst performing model. All top performing models consist of 8 to 12 factors.

In Table 3, number of occurrences of each factor with $\#Occur > 20$ is presented. Out of 19 factors, 4 occurred in all top performing models, namely, betweenness centrality, maximum cyclomatic complexity, sum of cyclomatic complexity and maximum LOC. Furthermore, 3 factors occurred in more than 90% of the top performing models namely fix count, fix count rank and sum of LOC.

When compared with the significance of factors in univariate regression analysis, code based measures performed surprisingly well in the top models. Even though we could not validate hypotheses H_4 and H_5 during univariate analysis, cyclomatic complexity and LOC measures of methods related with reopened issues have significant benefits to predictive model. On the other hand, betweenness is significant both in the best models and in the univariate regression model.

3.3 Multivariate Statistical Model

In Figure 3, AUC for the model that performed the best (in predicting reopened issues) during our exhaustive search is presented. In addition to the 0.81 AUC, the top model had 0.88 recall and 0.82 precision. If this was a prediction scenario, we could conclude that the model had a significant potential. However, some of the factors such as factors related to code issue relation have limited applicability in a prediction scenario because they are only available post-mortem.

Table 3: Number of Occurrences of Factors In Top Performing Model Combinations

Factor Name	Occurrence In 100 Best Model (%)
Same Location	100
Betweenness Centrality	100
Maximum Cyclomatic Complexity	100
Sum of Cyclomatic Complexity	100
Maximum LOC	100
# Fixed Issues	94
Sum of LOC	94
Rank in Fixed Issue Count	91
Unique Methods Changed	59
Total # Edits	44
Degree	36
Degree Centrality	24
Duration Between First and Last Fix	24

The factors in Table 3 with the highest number of occurrences in top 100 models were present in the top model. The model with the highest AUC contains the following factors:

Same Location - Betweenness Centrality - Maximum Cyclomatic Complexity - Sum of Cyclomatic Complexity - Maximum LOC - # Fixed Issues - # Sum of LOC - Rank in Fixed Issue Count - Unique Methods Changed - Degree - Duration Between First and Last Fix - Total # Edits

4. DISCUSSION

4.1 Interpretation of Results with QA Team

In order to interpret our findings, we held a meeting with the QA Team in the company and asked free format questions about our analysis. Responses are summarized below.

Betweenness centrality: Why do you think that reopened issues are often located at the centre of issue proximity network?

Reopened issues are generally the ones that developers postpone fixing or cancel, since a) they may decide that it is not very critical for the customer and b) fixing it may be risky or complex so that they may want to delay fixing this issue.

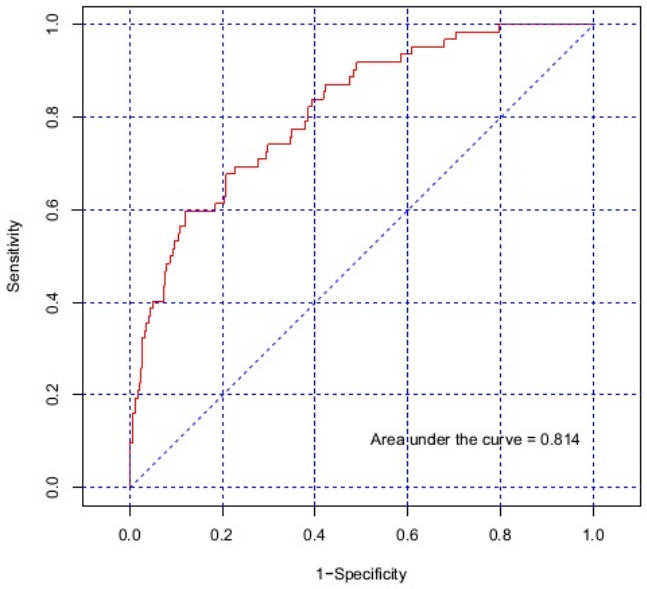

Figure 3: ROC curve of the multivariate regression that uses 12 factors which performed the best in model testing - AUC : 0.81 (Recall: 0.88 and Precision: 0.82)

The fact that reopened issues are at the center of this network supports our anecdotal evidences of issues' complexity.

Rank of developers in terms of method changes: Do you think that developers who edit many methods should have a positive or negative impact on reopened issues?

Usually, the amount of code changed by a developer is positively correlated with the amount of issues fixed by this developer. These top "code-altering" developers may decide to postpone or cancel some of these issues assigned to them, until they reduce their workload, since fixing a new issue may also introduce others due to large amount of changes.

4.2 Threats to Validity

In this section, we discuss possible threats to validity of our study. We have used a large scale enterprise product to conduct a case study. Even though drawing general conclusions from an empirical study is very difficult, results should be transferable to other researchers with well-designed and controlled experiments. In this study, we propose a set of metrics representing four main dimensions to investigate their effects on reopened issues. Our methodology consists of investigating their individual effects on reopened issues, as well as pairwise metric relations to find the best set of metrics predicting reopened issues. Using the same methodology, results can be replicated and refuted on new datasets.

In this case study, we were able to extract 2287 resolved issues of which 1046 were matched with the code base and developers. At a first glance, this seems to be a small set, however we traced all issues submitted since 2004 and filtered them based on our requirements. While reducing the dataset, we also considered the fact that ratio of reopened issues over other issues should be similar to the ratio of the initial dataset. When we linked issues with developer activities, this ratio was 7%, whereas in the final set, this ratio was reduced to 6%, with a minor change. We have also worked closely with QA team in the company to validate data quality.

We use logistic regression during univariate and multivariate analyses, since coefficients in logistic regression are easily interpretable with their significance regions. We did not consider using other algorithms and compre their performance with regression, since our aim was to understand the explanatory power of metrics on reopened issues, rather than building the best predictive model for reopened issues. But, we did considered all factor combinations to select the best metric set during our experiments.

5. RELATED WORK

Issue management systems of large open source software projects have become available on the internet since early 1990s. Issue management systems of some applications with commercial licences are publicly accessible since the owners of these applications would like to make their issue handling process transparent to their users. Research on issue repository data has started in parallel with the public availability of past software issue management data.

There are two recent papers closely related to our research. Shihab et al. [19] analysed work habits, bug report, bug fix dimensions for Eclipse Project to find the factors that contributed to bug reopening and built a *reopened bug prediction model* using decision trees. Shihab et al. found that *the comment text, description text, time to resolve the bug and the component the bug was found* were the leading factors that caused bug reopening for Eclipse [19]. On the other hand, Zimmermann et al. [25] analysed Windows Vista and 7 issue repository and conducted a survey on 394 developers in order the important factors that causes bug reopens [25]. Zimmermann et al. built a logistic regression model in order to identify the factors that may cause issue reopening. In their research Zimmermann et al. used organizational and process related factors in addition to factors directly extractable from the issue report. In their logistic regression model nearly all the factors they observed were found to be significant which included factors related to location, work habits and bug report characteristics.

Our work is different than the research by Shihab et al. and Zimmermann et al. in two aspects: 1) The factors and the dataset we analysed are different, 2) We analysed the effect of the combinations of various factors in addition to individual factors on issue reopening.

Other notable recent areas of research include automated issue triage, factors that change the quality of issue reports, detection of duplicate issues and estimation of issue fix durations [3, 6, 21].

One important related research topic about issues is automated *bug triage*, the procedure of processing an issue report and assigning the right issue to the right developer. This problem is especially important for large software with millions of users. Anvik et al. found that 300 daily reported issues make it impossible for developers to triage issues effectively for Mozilla based on an interview with an anonymous developer [3, 12, 21].

Text mining methods have been used in several studies to find the most relevant developer to handle a bug in auto-

mated bug triage models [2–4, 9, 20]. Bakir et al. proposed a model that forwarded auto-generated software fault data directly to the relevant developers by mining the patterns in the faults [5]. The benefit of automated bug triage is often measured by % of actual owners estimated by the model and the decrease in issue reassignments or *bug tosses*. While bug triage studies claim that bug tossing is time consuming, [20], Guo et al. observed that issue reassignment is beneficial for communication [13].

Effective issue reporting is also important for reporters as for developers. In an exploratory study on Windows Vista, Guo et al. [12] identified the characteristics of bugs that are getting fixed. Bettenburg et al. [6] also analysed the components of a bug report (severity, stack traces, builds, screenshots) that make a bug more likely to be resolved. Estimation of issue resolve times is another research area to plan developers' efforts efficiently. Some studies on open source projects can be found in ([11], [21]).

6. CONCLUSIONS

In this paper, we analysed the effects of 19 factors from four different dimensions on the probability of issue reopening for a large-scale software developed in geographically distributed locations. In our study, we have found that a subset of these factors are important for issue reopening. The predictive power of best factor combinations is high with $AUC \approx 0.81$ in the best performing models.

RQ: Which factors lead to issues getting reopened?

In order to find the factors that were most important for issue reopening, we built a univariate and best-subset logistic regression models. In the univariate logistic regression model and the best-subset logistic regression model we checked the importance of the factors we considered.

Dimensions of developer activity (in terms of the time between first and last issue fixes and the number of methods edited during issue fixes), issue proximity network (in terms of common methods changed during issue fixes) and geographical locations of issue owners and originators) are found to be important for issue reopening. In the top ranking logistic regression models based on their predictive power, factors from *all* dimensions were prominent.

In previous research on this topic, nearly all of the considered factors were found to be significant [19], [25]. On the contrary, in our analysis we found that a subset of our considered factors are significantly more important in issue reopening. The best logistic regression model in terms of predictive power contains 12 factors out of 19 (Section 3.3 for the full list).

Implications of the Results To The Industry

Issue reopening can lead to unanticipated resource allocation, leading to projects running over budget and late. Therefore, it is important to proactively identify issue that can be reopened and take corrective actions. Quantitative findings of our study suggest that issue complexity and developers workload play an important role in triggering issue reopening. This information can aid managers in deriving concrete corrective actions (e.g., ensuring existence of deep code review and reducing developer's workload.

As indicated in our results, issue reopening may have many reasons and may not be modelled by a small set of factors. In addition, some of the causes of the issue reopening such as design problems may be outside the scope of the issue management process. Identifying, the important factors that may lead to issue reopening may be the first step to lead companies to understand these underlying causes and take necessary actions.

Table 4: A list of phases in which issues were found and reported.

Phase	Issues reported during this phase (%)
Customer	25.5
Functional testing	19.9
Regression testing	8.5
Coding	7.4
System testing	6.8
Nightly build	5.0
Performance testing	3.9
Design	3.6
Unit testing	2.7
Beta testing	1.3
Others	15.5

Future Work

Every model is a simplification of reality and has its limitations. We attempted to model the 3 aspects of software development (people, process, product) when choosing the factors. New factors can be proposed in the future studies related to these aspects. As usual, in this case study, one possible future work would be testing our conclusions in new datasets. Another area to consider would be analysing the causality relations between the considered factors and the probability of issue reopening.

APPENDIX

Table 4 presents majority of phases in which issues were found and reported in the company and percentage of their occurrence. We have listed 10 phases which account for 85% of all issue reports, and defined other phases as "Others" due to privacy issues.

Acknowledgment

This research is supported in part by Turkish State Planning Organization (DPT) under the project number 2007K120610 and partially supported by TEKES under Cloud-SW project in Finland. We would like to thank IBM Canada Lab – Toronto site for making their development data available for research and strategic help during all phases of this research. The opinions expressed in this paper are those of the authors and not necessarily of IBM Corporation.

7. REFERENCES

[1] E. Alpaydin. *Introduction to Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2004.

[2] J. Anvik, L. Hiew, and G. C. Murphy. Who should fix this bug? In *Proceedings of the International Conference on Software Engineering*, pages 361–370, Shanghai, China, 2006.

[3] J. Anvik and G. Murphy. Determining implementation expertise from bug reports. In *Mining Software Repositories, 2007. ICSE Workshops MSR'07. Fourth International Workshop on*, pages 1–8. IEEE, 2007.

[4] J. Anvik and G. C. Murphy. Reducing the effort of bug report triage. *ACM Transactions on Software Engineering and Methodology*, 20(3):1–35, Aug. 2011.

[5] A. Bakir, E. Kocaguneli, A. Tosun, A. Bener, and B. Turhan. Xiruxe: An Intelligent Fault Tracking Tool. *AIPR09, Orlando*, 2009.

[6] N. Bettenburg and A. Hassan. Studying the Impact of Social Structures on Software Quality. In *2010 IEEE 18th International Conference on Program Comprehension*, pages 124–133. IEEE, 2010.

[7] L. Briand, W. Melo, and J. Wust. Assessing the applicability of fault-proneness models across object-oriented software projects. *Software Engineering, IEEE Transactions on*, 28(7):706–720, 2002.

[8] B. Caglayan, A. Tosun, A. Miranskyy, A. Bener, and N. Ruffolo. Usage of multiple prediction models based on defect categories. In *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, pages 1–9. ACM, 2010.

[9] D. Cubranic and G. Murphy. Automatic bug triage using text categorization. In *Proceedings of the Sixteenth International Conference on Software Engineering Knowledge Engineering*, pages 1–6. Citeseer, 2004.

[10] A. Gelman and J. Hill. *Data Analysis Using Regression And Multilevel/Hierarchical Models*. Analytical Methods for Social Research. Cambridge University Press, 2007.

[11] E. Giger, M. Pinzger, and H. Gall. Predicting the Fix Time of Bugs. In *RSSE '10 Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering*, pages 52–56, 2010.

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

[13] P. Guo, T. Zimmermann, N. Nagappan, and B. Murphy. Not my bug! and other reasons for software bug report reassignments. In *Proceedings of the ACM 2011 conference on Computer supported cooperative work*, pages 395–404. ACM, 2011.

[14] J. Herbsleb and A. Mockus. An empirical study of speed and communication in globally distributed software development. *IEEE Transactions on Software Engineering*, 29(6):481–494, June 2003.

[15] S. Koch. Effort modeling and programmer participation in open source software projects. *Information Economics and Policy*, 20(4):345–355, Dec. 2008.

[16] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *Software Engineering, IEEE Transactions on*, 34(4):485–496, 2008.

[17] C. d. Mazancourt and V. Calcagno. glmulti: An r package for easy automated model selection with (generalized) linear models. *Journal of Statistical Software*, 34(i12), 2010.

[18] A. T. Misirli, B. Caglayan, A. V. Miranskyy, A. Bener, and N. Ruffolo. Different strokes for different folks: a case study on software metrics for different defect categories. In *Proceedings of the 2nd International Workshop on Emerging Trends in Software Metrics*, WETSoM '11, pages 45–51, New York, NY, USA, 2011. ACM.

[19] E. Shihab, A. Ihara, Y. Kamei, W. M. Ibrahim, M. Ohira, B. Adams, A. E. Hassan, and K.-i. Matsumoto. Predicting Re-opened Bugs: A Case Study on the Eclipse Project. *2010 17th Working Conference on Reverse Engineering*, pages 249–258, Oct. 2010.

[20] A. Tamrawi, T. Nguyen, and J. Al-Kofahi. Fuzzy set-based automatic bug triaging: NIER track. *Proceedings of the 33rd International Conference on Software Engineering*, pages 884–887, 2011.

[21] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller. How long will it take to fix this Bug? In *Fourth International Workshop on Mining Software Repositories, 2007. ICSE Workshops MSR'07*, number 2, 2007.

[22] E. J. Weyuker, T. J. Ostrand, and R. M. Bell. Using developer information as a factor for fault prediction. In *Proceedings of the Third International Workshop on Predictor Models in Software Engineering*. IEEE Computer Society, May 2007.

[23] S. Zaman, B. Adams, and A. E. Hassan. Security Versus Performance Bugs : A Case Study on Firefox. *Design*, pages 93–102, 2011.

[24] T. Zimmermann and N. Nagappan. Predicting defects with program dependencies. *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pages 435–438, Oct. 2009.

[25] T. Zimmermann, N. Nagappan, P. Guo, and B. Murphy. Characterizing and predicting which bugs get reopened. In *Proceedings of the 34th International Conference on Software Engineering [ACCEPTED]*, 2012.