Improving Bug Triaging with High Confidence Predictions at Ericsson

Aindrila Sarkar
Department of Computer Science
and Software Engineering
Concordia University
Montreal, Canada
Email: to.aindrila1989@gmail.com

Peter C. Rigby
Department of Computer Science
and Software Engineering
Concordia University
Montreal, Canada
Email: peter.rigby@concordia.ca

Béla Bartalos
Ericsson
Budapest, Hungary
Email: bela.bartalos@ericsson.com

***Abstract*—Correctly assigning bugs to the right developer or team, *i.e.* bug triaging, is a costly activity. A concerted effort at Ericsson has been done to adopt automated bug triaging to reduce development costs. In this work, we replicate the research approaches that have been widely used in the literature. We apply them on over 10k bug reports for 9 large products at Ericsson. We find that a logistic regression classifier including the simple textual and categorical attributes of the bug reports has the highest precision and recall of 78.09% and 79.00%, respectively.**

Ericsson's bug reports often contain logs that have crash dumps and alarms. We add this information to the bug triage models. We find that this information does not improve the precision and recall of bug triaging in Ericsson's context.

Although our models perform as well as the best ones reported in the literature, a criticism of bug triaging at Ericsson is that the accuracy is not sufficient for regular use. We develop a novel approach where we only triage bugs when the model has high confidence in the triage prediction. We find that we improve the accuracy to 90%, but we can make predictions for 62% of the bug reports.

***Index Terms*—Bug Triaging, Machine Learning, Log Analysis, Incremental Learning**

I. Introduction

Bug fixing is an integral part of development and maintenance phase in the lifecycle of a software project. Large critical software projects must deal with customer bugs quickly. The first step in the process is to triage the bug by assigning it to the team or developer that can fix the bug. The large volume of bug reports submitted daily makes manual bug triaging a time-consuming process. Furthermore, when a bug is assigned to the wrong team or developer, the cost and time to fix the bug is increased. We conduct a case study at Ericsson, which has a significant number of internal and external bug reports submitted daily. Ericsson uses a bug tracking system developed at Ericsson. The first level of triage involves routing the bug reports to an appropriate team. Human triagers do this job manually.

In this paper, we focus on automation of assigning the bug reports to the correct team. There is a large and diverse related work including approaches based on machine learning [3] [8] [18] [14], social network analysis [26] [31] [12], experience model of developers [24] [30] [21] [28], and mining software repository [20] [16] [19] [22]. The majority of previous works address bug assignment to individual developers. In contrast, at Ericsson, bugs are assigned to the development teams not the developers and we assign bugs to 43 teams.

We apply a simple machine learning approach to assign bug reports at Ericsson. Selection of the features is very important for supervised machine learning. The majority of triaging techniques in the past use text descriptions of bug reports [3] [8] [6] and categorical attributes, including product, component, severity [18] [5] [14] [11]. In our work, we find that right selection of categorical features combined with textual contents is effective. We also explore additional information from the alarm logs and crash dumps attached to the bug reports. Furthermore, unlike many works in the past that use cross validation or a small test dataset for evaluation, we use more realistic time split validation to evaluate our techniques.

Although our models have a comparable accuracy to other large scale bug triaging research works, they were not sufficient for regular use at Ericsson. While we can suggest the top N developers and improve our accuracy, this approach still requires manual triage effort to decide among the top suggestions. Instead, we decide to only triage those bug reports for which the model has high prediction confidence. Using this approach we are able to attain a high accuracy and reduce the manual effort by over half.

RQ1. Replication: How well do existing bug triaging approaches, which contain textual and categorical features, work at Ericsson?

The most common bug triage models contain the texts of the bug report, *e.g.,* summary, description and categorical features, *e.g.,* product, severity. We find that a model with these simple attributes has a precision and recall of 78.09% and 79%, respectively. The categorical features have the strongest predictive power.

RQ2. Alarms and Crash Dumps: Does the information contained in alarm logs and crash dumps help in bug triaging?

Bug reports often contain crash dumps and other log information. We make a novel contribution by determining how well this data helps in bug triaging. Alarm logs and crash dumps are available for only 51.17% and 5.52% of all bug reports. Even when we train the models with the subset of bug reports that contain the alarms, there is no improvement

in the precision and recall when compared with the model that contains only the textual and categorical features.

RQ3. High Confidence Predictions: What is the impact of high confidence prediction on the accuracy of triaging?

At Ericsson, even the highest precision and recall values were too low to be used in production. Previous works have dealt with this issue by suggesting the top N developers. Although we have a top 3 accuracy above 90%, Ericsson did not like this approach because it still requires manual intervention. Jonsson et al. [15] first introduce the concept of using prediction confidence for identifying the faulty components from the bug reports. We introduce this novel approach in the context of bug triaging by assigning the bug reports when the model has high confidence in the prediction. We find that when the model is 90% confident in the result, we are able to triage the bug reports with a precision and recall of 89.75% and 90.17%, respectively. However, we only triage 61.71% of the total bug reports.

This paper is structured as follows. In Section II, we describe the existing bug triaging models. In Section III we explain the testing and triaging process of Ericsson. In Section III we also describe our case study dataset and the methodology of our approach. In Section IV, V, VI we present the results. In Section VII we describe the threats to validity. Finally in Section VIII we conclude our work.

II. Related Work

In this section, we discuss the related work on the basis of the recommendation technique, different attributes of the bug reports used to train the model, evaluation metric, results and evaluation setup. We present the results of these previous works for their top1 recommendation. Although researchers have also reported the results for topN recommendation. Table I provides a summary of the related works.

The fixer recommendation techniques used in the previous works explore a wide range of approaches including machine learning (ML), bug tossing graph models, mining software repositories (MSR), social network analysis and developers' activity models. We divide the previous works in these 5 categories. Information retrieval (IR) techniques have been widely used in almost all the categories. A significant number of researchers use the common IR approach of Term Frequency multiplied by inverse document frequency (TF-IDF) to vectorize the texts. Some authors have also used other IR techniques including Latent Semantic Indexing (LSI) to reduce the dimensions of term vectors. Latent Dirichlet Allocation (LDA), a popular topic modelling algorithm has also been used by many authors. Some researchers have also used natural language processing (NLP) techniques to mine informative terms from the texts.

A. ML techniques

Previous works investigate different techniques with a majority of them analyzing textual information of the bug reports. A wide variety of classifiers including Decision Tree, SVM, Naive Bayes and ensemble classifier has been used in the previous works. Recent works also investigate the use of deep learning techniques such as convolutional neural network (CNN) with Word2Vec as the word embedding technique.

The early work by Anvik et al. [3] vectorize the text of the summary and description by normalized TF-IDF and use Naive Bayes, SVM and C4.5 to identify an appropriate fixer. In a later work [4], they generalize beyond a single fixer to recommend components and other potential developers. They achieve a maximum precision and recall of 64% and 10% respectively.

Lin et al. [18] perform an empirical study on bug assignment in industrial projects. They vectorize the textual contents with TF-IDF and train a SVM classifier. They also explore the categorical fields of the bug reports and use decision tree which outperforms SVN. They achieve an accuracy of 77.64%.

Banitaan et al. [5] propose an approach by using both of text and categorical attributes. They use traditional TF-IDF and Chi Square for feature selection. Using Naive Bayes they achieve a precision and recall of 66.6% and 63.8% respectively.

Canfora et al. [8] use probabilistic textual similarity for change request (CR) assignment. For every developer they build a descriptor (*i.e.* vector of terms) using short and long description of the CRs that the developer has fixed. The probabilistic model compares the new CR descriptor to developer descriptors to recommend the fixers. They achieve a recall of 59%.

Ahsan et al. [1] investigate the use of the LSI for reduction of dimensions of term vectors. They combine this with various classifiers and get the best results with SVM. The accuracy, precision and recall they achieve are 44.4%, 37% and 35% respectively.

Jonsson et al. [14] use ensemble learning to combine the outcome of multiple classifier in a single recommender. To train the individual classifier, they use the textual contents and categorical attributes of the bug reports of industrial projects. The best accuracy they achieve using 10 fold cross validation is 85%. Although, when they evaluate using time split validation, best accuracy is 65%.

Florea et al. [11] develop a spark based fixer recommender system. Using NLP they only preserve nouns from the texts and use TF-IDF for vectorization. They also use other attributes including product and component. SVM with liblinear outperforms others on their dataset. They achieve a precision and recall of 89% and 88% evaluating on a small test dataset.

Lee et al. [17] use convolutional neural network (CNN) with Word2Vec on text. The highest accuracy they achieve is 85% for industrial projects and 46% for open source projects. Chen et al. [9] extend this work on incident triaging and perform a comparative study among deep learning, supervised classifiers, KNN, topic modelling, tossing graph and fuzzy based techniques. The highest accuracy they achieve using deep learning is 71%.

B. Bug Tossing

The reassignment of a bug that has been incorrectly triaged first time, *i.e.* bug tossing, has been studied by some re-

TABLE I
COMPARISON OF PREVIOUS WORKS: TECHNIQUES, ATTRIBUTES, EVALUATION METHODS, AND RESULTS

Author	Technique		Attributes Used			Best Result				Evaluation
	ML based	Others	Textual	Nominal	Others	Accuracy	Precision	Recall	F Score	
Anvik et al. [3]	SVM with TF-IDF	-	Summary, Description	-	-	-	64% (top1)	10% (top1)	-	On fixed test data
Lin et al. [18]	SVN, Decision Tree	-	Summary, Description	Module, Phase, Priority	-	77.64% (top1)	-	-	-	10 fold cross validation
Banitaan et al. [5]	Naive Bayes with TF-IDF & Chi-Square	-	Summary, Description	Reporter, Component	-	-	66.6% (top1)	63.8% (top1)	-	5 fold cross validation
Canfora et al. [8]	-	Probabilistic IR	Summary	-	-	-	-	59% (top1)	-	On fixed test data
Ahsan et al. [1]	SVM with TF-IDF & LSI	-	Summary, Description	-	-	44.4% (top1)	37% (top1)	35% (top1)	-	On fixed test data
Jonsson et al. [14]	Ensemble stacked generalizer	-	Summary, Description	Submitter, Site, Revision, Priority	-	85% (top1) 65% (top1)	-	-	-	10 fold cross, Time split validation
Florea et al. [11]	SVM with Liblinear, TF-IDF & chi-square	-	Summary, Description	Product, Component	-	-	89% (top1)	88% (top1)	-	On small test data
Jeong et al. [13]	-	ML with Bug Tossing Graph	Summary, Description	-	-	77.14% (top5)	-	-	-	On fixed test data
Bhattacharya et al. [7]	-	ML with multi feature bug tossing	Summary, Description	Product, Component	-	38.03% (top1)	-	-	-	Time split validation
Matter et al. [20]	-	Similarity of vocabulary by IR	-	-	source code, commits	34% (top1)	-	-	-	On fixed test data
Kagdi et al. [16]	-	MSR for source codes predicted by IR	Description	-	Source code, Commits	94% (top1)	-	-	-	On small test set (18)
Shokripour et al. [23]	-	MSR for files predicted by phrase similarity	Summary, Description	-	-	-	-	31% (top1)	-	-
Shokripour et al. [22]	-	MSR for source codes predicted by NLP	Summary, Description	-	source codes, commits	48.23% (top1)	-	-	-	On fixed test data
Linares et al. [19]	-	Authors of source codes predicted by IR techniques	Summary, Description	-	source codes	-	63%	64%	-	On fixed test data
Zhang et al. [31]	-	ML & social network analysis	Summary, Description, Comments	Components	-	43.98% (top1)	-	-	-	Time split validation
Hu et al. [12]	-	Social network of developer, code component & bug	Summary, Description	-	Commits, Change sets	-	-	42.36% (top1)	-	Time split validation
Zhang et al. [31]	-	IR & social network analysis	Summary, Description, Comments	-	-	-	-	-	25% (top 1)	Time split validation
Tamrawi et al. [24]	-	Developer's expertise via fuzzy set	Summary, Description	-	-	51.2% (top1)	-	-	-	Time split validation
Wang et al. [25]	-	Caching developer's component level activities	-	Component	-	54.32% (top1)	-	-	-	Time split evaluation
Zhang et al. [30]	-	Topic modelling & correlation with bug reporter	Summary, Description	Reporter	-	-	-	-	71% (top1)	On fixed test data
Naguib et al. [21]	-	Topic Modelling & developer's activity profile	Summary, Description	Component	-	30% (top 1)	-	-	-	On fixed test data
Yang et al. [28]	-	Using topic modelling & developer's expertise score	Summary, Description	Product, Component, Priority	-	63% (top1)	-	-	-	On fixed test data
Xia et al. [27]	-	Multi feature topic modelling	Summary, Description	Product, Component	-	68.68% (top1)	-	-	-	Time split validation
Lee et al. [17]	-	CNN with Word2Vec	Summary, Description	-	-	85% (top 1)	-	-	-	- Fixed test data
Chen et al. [9]	SVM, KNN, NB, CNN	Topic Modelling, Tossing Graph, Fuzzy Set	Summary, Description	-	-	71% (top 1)	-	-	-	Fixed test data
Sarkar et al.	Logistic regression with TF-IDF	-	Summary, Description	Multiple *e.g.,* Product	Machine dumps	-	78.09% (top 1)	79.00% (top1)	-	Time split validation

searchers.

Jeong et al. [13] introduce the idea of using markov model based bug tossing graphs to recommend fixers of a bug. They use Naive Bayes and Bayesian Networks with TF-IDF and integrate tossing graph information into the prediction of the classifiers. The best accuracy they achieve is 77.14% for top5 recommendation. Bhattacharya et al. [7] extend this work by using categorical features including product and component of the bug reports. They use these features to train the classifier and also into the tossing graph. For top 1 recommendation, they achieve an accuracy of 38.03%

C. MSR based techniques

Software repositories, such as source code and version tracking systems, contain important historical information of how a system was developed and maintained. Researchers have mined this information to help in bug assignment.

Matter et al. [20] introduce an IR and vocabulary based approach. They recommend developers whose commit vocabulary is most similar to the vocabulary of the bug reports by comparing the term frequencies using IR technique. They achieve a precision of 34%.

Kagdi et al. [16] use identifiers (*e.g.,* class, methods) and comments from the source code and create a corpus for every source code file. The corpus indexed by LSI is then used to compute the similarity with the bug descriptions to predict the files related to the bug. They recommend developers based on their activities with these files in the version repository. They achieve an accuracy of 94% on a small test dataset of 18 bug reports.

Shokripour et al. [22] use NLP to mine the nouns in commit messages, comments in the source code, and previously fixed bug reports. Files related to a new bug report are predicted using a term weighting scheme. Developers are recommended based on their expertise with the predicted files. They achieve an accuracy of 48%.

In another work, Shokripour et al. [23] use the phrase compositions *i.e.* a NLP technique from the comments of the commits and the bug descriptions. They recommend the developers that are most active on the files with the most similar phrase compositions. They achieve an accuracy of 31%.

Linares et al. [19] use code authorship information, identifiers and comments of the source code file. Similarity of the corpus indexed by LSI is computed between the files and the bug report description with the author of top N most similar files being recommended. The best precision and recall they achieve is 63% and 64% respectively.

D. Social Network Analysis

Developers often collaborate with each other in the bug resolution process. Social network analysis originated from sociology has attracted some researchers to model the bug assignment problems using a network of developers as nodes and their collaborations as edges.

Zhang et al. [31] combine social network analysis with machine learning. Developers' contribution score determined by fixing, commenting, reporting bugs is added to the classifier score to recommend the developers. They achieve an accuracy of 43%.

Hu et al. [12] propose BugFixer which computes the similarity with other bug reports and recommends the developers by constructing a network with the associations among the developers, components, and bugs. They achieve a recall of 42%.

Zhang et al. [29] use IR techniques to find the similar bugs and recommend the developers based on fixing probability determined by the social network technique and fixing experience computed by the number of bug reports fixed and assigned by the developer. They achieve an F score of 25%.

E. Activity Models

In recent years, researchers have focused on specialized techniques that model developer's expertise by their activities including fixing, commenting and reporting the bugs.

Tamarawi et al. [24] develop a fuzzy set and cache based tool called Bugzie which maintains a fuzzy set of developers for every technical term. It predicts the developers by modelling the fixing correlation of developers with the technical terms based on their fixing activities in the past. It recommends the developers who recently participated in bug resolution (*i.e.* the developers in the cache). They achieve an accuracy of 51.2%.

Wang et al. [25] introduce FixerCache which caches developers based on their bug fixing activity at component level for a certain period and recommends the fixers by their activeness score in the cache. They achieve an accuracy of 54.32%.

Zhang et al. [30] propose an approach that combines topic models and the relationship between the bug reporter and the fixer. They recommend the developers based on the correlation score of a developer with a topic and an active reporter. They achieve an F Score of 71%.

Naguib et al. [21] propose an approach leveraging topic modelling and the developers' activities, including review, fixing and assigning bug reports. They recommend the developers based on the association scores towards the topics of the bug reports determined by these activities. The best accuracy they achieve is approximately 30%.

Yang et al. [28] propose a new method by introducing topic model and multiple feature including product, component, severity, priority. They extract the set of candidate developers who have contributed to the bug reports having same topic and the features and rank them by the scores determined using their number of activities *e.g.,* commits, comments, bug assignment. They achieve an accuracy of 63%.

Xia et al. [27] extend the basic topic modelling algorithm LDA and propose multi feature LDA that includes product and components. They recommend the developers based on the affinity scores of a developer towards a topic and the feature combination. They achieve an accuracy of 68%.

III. Case Study Data and Methodology

Ericsson develops and maintains large and critical software projects. Figure 1 shows where bug reports originate from: internal testing including integration, validation and performance testing and the customers. After code is committed, it runs through multiple levels of testing from low level unit tests run by developers to expensive simulations of real world scenarios on hardware. When a test fails, testers investigate whether it is an environmental problem or a product fault. If it is determined to be a product fault, a bug report is created and triaged. Ericsson customers are large telecom providers and in this work, there are bug reports from over 300 customers. At Ericsson, the triage process is done at the team level instead of individual developers.

We collect all the bug reports with status FIXED and reported between July 2016 to the end of June 2018. We keep the duplicate bug reports in our dataset as previous work [6] has shown that duplicate bug reports are useful. Our dataset contains 11,570 bug reports, fixed by 43 teams, across 9 products. There are 301 customers reporting the bugs. 51% of bug reports contain alarm logs and 5.5% of bug reports contain crash dumps. Figure 2 shows the number of bugs fixed by each development team. The data set is quite skewed with 81% of the reports being fixed by 8 teams.

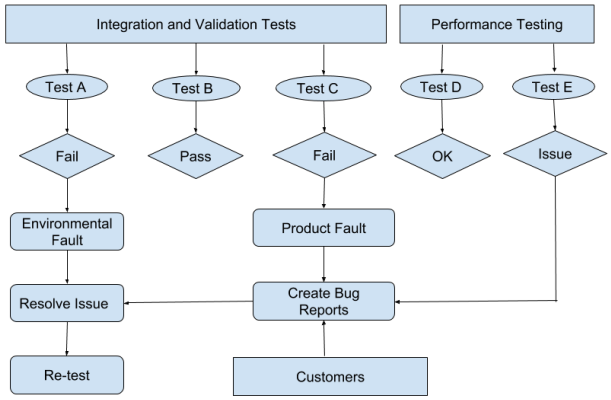

Fig. 1. Bug report process

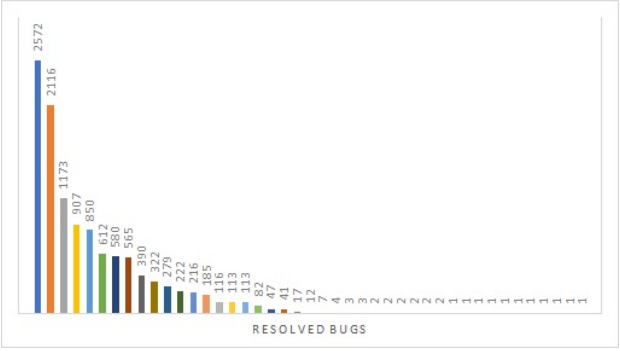

Fig. 2. Distribution of bug reports across the 43 teams

The data attributes we extract from the bug reports are summarized in Table II. The attributes are of three types - **Textual Attributes** *e.g.,* summary, description and answer **Categorical Attributes** *e.g.,* product, customer, site, submitter, priority, configuration and generation of product **Log Attributes** *e.g.,* alarms and crash dumps

TABLE II
Bug report features used in the models

Type	Feature	Description
Text	Summary	A short description of the issue
Text	Description	Detailed description of the issue. *e.g.,* Configuration of the hardware, steps to reproduce the issue etc.
Text	Answer	Description of the final solution made to fix the issue
Categorical	Product	Product that the issue was found on
Categorical	Customer	Customer that uses the product the issue was found on
Categorical	Site	Location the issue was reported from *e.g.,* Ottawa-Canada, Kista-Sweden
Categorical	Submitter	Team that reported the issue
Categorical	Priority	Priority of the issue. *e.g.,* Major, Medium, Minor
Categorical	Configuration	Higher level category of the product. *e.g.,* Baseband, Radio Software etc.
Categorical	Generation	Generation of the product. *e.g.,* Gen1, Gen2 etc.
Log	Alarms	Software and hardware errors or warnings that can be potential threat for the system
Log	Crash Errors	Errors generated by a program or processor crash

A. Textual Attributes

We extract the textual data from the summary, description, and answer of the bug reports. The answer is written by the developer at the time of closing the bug report, so we use this field in training data set only, not in the validation data set.

Preprocessing: We apply the standard preprocessing steps on the textual contents including tokenization, stop words removal, and stemming. Tokenization splits the text into multiple tokens. By applying stemming, the words are converted into their root forms. Stop words are the frequently used insignificant words which are removed. The terms which are very rare and appear in very few documents are also removed.

Feature Extraction: In most of the works focusing on prediction of bug report assignee using machine learning, conventional TF-IDF term weighting scheme has been used to vectorize the texts [11] [14] [3]. After preprocessing we apply normalized TF-IDF to the text contents.

Definition 1. $TF-IDF_{t_i,b_j} = tf_{t_i b_j} \times \left(1 + log\left(\frac{1+B}{1+b(t_i)}\right)\right)$

B denotes the total number of bug reports, $tf_{t_i b_j}$ is the number of occurrences of term t_i in bug report b_j and $b(t_i)$ is the number of bug reports in which term t_i has occurred.

B. Categorical Attributes

Along with text attributes, categorical attributes of the bug reports play a very important role in bug triaging. Categorical

features have been widely used in the context of bug report assignee recommendation [18] [5] [14] [11]. We use the following categorical features: 1. Product 2. Customer 3. Site 4. Priority 5. Submitter 6. Configuration 7. Generation of the product.

Feature Extraction: Using one hot encoding we convert the categorical features into a binary feature vector to train the classifier. In one hot encoding, a categorical attribute of every data sample containing a particular value is represented by a binary vector. The length of the binary vector is the number of all possible values of a categorical attribute. The binary vector of a data sample contains only one in the position of the value that the data sample holds for the particular categorical attribute and zeros in the position of the remained of the values of the categorical attribute.

C. Alarms

Alarms are software and hardware errors and warnings that can be a potential threat for the system. At Ericsson they use a log processing tool to extract the alarms occurred on the digital and radio units. The alarms listed in the dump are used by the testers to identify the most pertinent part of the potential problems.

Processing: Using the internal log processing tool, we extract the alarms. We also extract the crash date if there is any program crash. The machine dumps may contain logs of several days if the nodes are not cleared. We look for the alarms occurring on the day of crash. If there is no crash, we look for the alarms occurred on the last date in the log. For a specific problem there can be multiple causes. We concatenate the text of the problem and cause together to make it a single line. Then we select unique lines of problem and cause, removing the duplicate ones. Figure 3 illustrates this process.

Feature Extraction: Instead of applying TF-IDF at term level, we apply it at line level. As we select only the unique lines, line frequency in this case is either 0 or 1. That is why it is referred by Line-IDF [2].

D. Crash Dumps

Within the machine dump there is a postmortem log which is generated when there is a program or processor crash. These logs contain traces and errors with the timestamp and source code file name, the trace or error is generated from and the trace or error message. The file names and error messages have the potential to be a good feature for bug triaging as the development teams may have some affinity towards a particular type of source code or error messages.

Processing: Using an internal Ericsson tool, we process the postmortem logs and extract the file names and the error messages of the crash errors. First we clean the error messages by removing hexadecimal codes and all the non alphabetic characters and ensure that each error message is unique. Figure4 illustrates this process.

Feature Extraction: Error messages contain multiple terms. Instead of using the terms as the unit of analysis, we apply the weighting scheme Line-IDF [2] at the line level. However for the source code file, TF-IDF is applied at term level as unlike error message they contain a single term.

Definition 2. $LINE - IDF_l = \left(1 + log\left(\frac{1+A}{1+A_l}\right)\right)$

A denotes the total number of logs. A_l denotes the number of logs that contain the log line l.

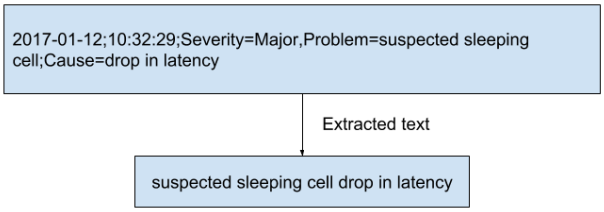

Fig. 3. Cleaning alarm logs

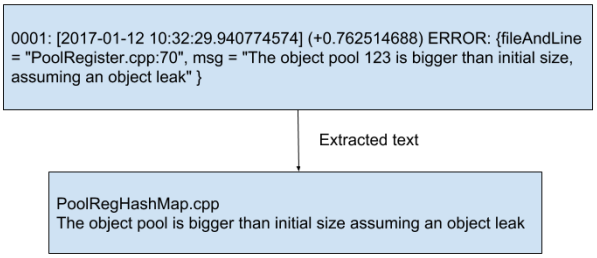

Fig. 4. Cleaning crash dumps

E. Classification With Logistic Regression Models

We use L2-regularized Logistic Regression with Liblinear solver [10] for classification. Logistic regression outperforms NaiveBayes, SVM, and KNN (see Section VII). For the best performing model, we vary the cutoff confidence score and analyze it's impact on the bug triaging accuracy. The classifier determines the probability for every class. Like a linear classifier, Logistic Regression Classifier multiplies the class specific weights (W_y) with the input features (X) and adds a bias (b) to calculate the class specific linear score (S_y). The linear score is then used to calculate the probability of the data sample belonging to each class. The class with the highest probability score is selected as the class decided by the classifier. We refer to the probability of the decided class by confidence.

Suppose there are k classes denoted by j = 1 to k. S_y is the linear score for the class Y and $P(Y|X)$ is the probability of the class Y.

$$S_y(X) = W_y^t X + b$$

$$P(Y|X) = \frac{exp(W_y^t X + b)}{\sum_{j=1}^{k} exp(W_j^t X + b)}$$

We tune the regularization parameter, C, of the logistic regression classifier. This parameter controls the overfitting.

The default value of C for the logistic regression classifier of scikit-learn library is 1.0. We ran the model varying $C = 1.0$ to $C = 10.0$ and find that after $C = 5.0$, the precision and recall vary by less than 0.30 percentage points. As a result, we report the values at $C = 5.0$. We report the class weighted average for precision and recall.

F. Evaluation Setup

To evaluate how well each attribute in the data helps in triaging bugs, we use a time split evaluation with an incremental learning framework that is common in the research literature [7] [25] [27]. We collect the bug reports of two years and sort them in chronological order. We split the dataset on a weekly basis W. We train on $W = 1$ to $W = T - 1$ and test on week T. Figure 5 illustrates our evaluation setup. We have experimented with alternative time frames, *e.g.,* months, and removing older data. We find that the difference in precision and recall is nominal with a decrease of .88 and .27 in precision and recall respectively (see Section VII). We create models to test each of our features independently as well as in the context of a model that combines multiple features. In total, we describe 8 models.

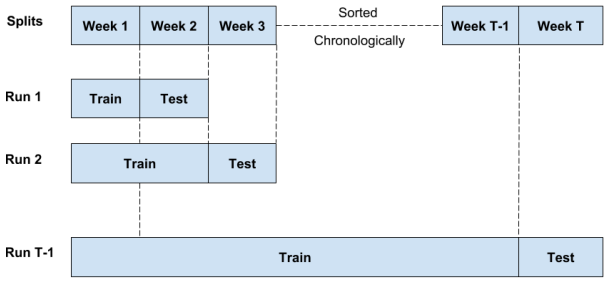

Fig. 5. Incremental evaluation setup

IV. Replication Results

RQ1: How well do existing bug triaging approaches, which contain textual and categorical features, work at Ericsson?

The simplest bug triaging approach uses a classification model with the textual descriptions contained in the bug reports worked on by each team or developer [3] [11]. Inclusion of categorical attributes such as the products, components is also an obvious method to triage bugs [18] [5]. For this research question, we implement existing traditional model to determine how well they work on Ericsson bug reports.

Model M1: Bug-Triaging model with only text: This simple model uses only text attributes of bug reports such as the summary and description and answers. We apply standard NLP preprocessing and use TF-IDF as the term weighting scheme (see Section III-A). In Table III, the text only model has a precision and recall of 62.49% and 64.13%, respectively.

Model M2: Bug-Triaging Model with only categorical features: We implement another simple model using categorical features including products, customers. In Table III, we see the model with categorical features outperforms the text model with a precision and recall of 73.70% and 74% respectively.

TABLE III
Model features and precision and recall

Model	Features	Precision	Recall
M1	Text	62.49	64.13
M2	Categorical	73.70	74.00
M3	**Text & Categorical**	**78.09**	**79.00**
M4	Text, Categorical, Alarms & Crashes	77.96	78.85
M5	Alarms	15.57	22.04
M6	Crashes	11.52	22.21

Model M3: Bug-Triaging Model with Text and Categorical: Combining the categorical and text features, we observe a precision and recall of 78.09% and 79%, respectively. In Table III, we see that this model outperforms M2 with an increase in precision and recall of 4.39 and 5 percentage points respectively.

> Model 3, which has both textual and categorical attributes, performs triaging of bug reports with a precision and recall of 78.09% and 79%, respectively. Categorical features have the strongest predictive power.

V. Results for Alarms and Crash Dumps

RQ2: Does the information contained in alarm logs and crash dumps help in bug triaging?

Bug reports often contain crash dumps and other log information. We make a novel contribution by determining how well this data helps in bug triaging. We use the internal Ericsson tools to extract alarm and crash details from the logs. The extracted information is textual and is processed using Line-IDF as described in Section III-D. This information is more difficult to extract than textual and categorical attributes of the bug reports. So, we create a combined model to determine how much additional predictive power this alarm and crash information add to existing models.

Model M4: Bug-Triaging Model with Text, Categorical, Alarms, and Crash features: This model includes all the attributes that we collect from the bug reports: text, categorical, alarms, and crash dumps. In Table III, we see that this model achieves a precision and recall of 77.96% and 78.85% respectively. Surprisingly this additional information does not improve the accuracy of bug triaging. Although text and categorical information are sufficient, for completeness, we create individual models for alarms and crash dumps to determine their independent ability to triage bugs.

Model M5: Bug-Triaging Model with only alarms: This model is trained with the alarms contained in logs. The alarm contains both problem and cause, so we use Line-IDF to capture the entire alarm text. In Table III, the alarm model achieves a precision and recall of 15.57% and 22.04% respectively. Across all bug reports, alarms are a poor feature for bug triaging.

Model M6: Bug-Triaging Model with only crash dumps: The model is trained with the crash error messages and the source code file names extracted from the crash dumps. We use Line-IDF for the crash error messages. In Table III, we see that this model achieves a precision and recall of 11.52% and 22.21% respectively. We observe that only 5.52% bug reports contain the crash dumps. The scarcity of bug reports with attached crash dumps and the low predictive power make crash dumps a poor feature for bug triaging.

Model M7 and M8: Bug reports that contain alarms: Half of the bug reports, 51.17%, contain alarms. We train two models on the bug reports that contain alarm logs. First we use the text and categorical features from our best model, M3, but train and test **Model M7** only on the bug reports that contain alarms logs. In Table IV, we see that this model achieves a precision and recall of 70.72% and 72.80% respectively.

Then we create **Model M8** that adds alarms to M7 to determine if these alarms improve the precision and recall on bug reports that contain alarms. In Table IV, we see model M8 achieves precision and recall of 70.09% and 72.28%. We see a slight reduction in precision and recall with M8 and conclude, that at Ericsson, the alarms do not provide information that improves the accuracy of bug triaging.

Performing same experiments as model M7 and M8 with crashes have not been possible due to scarcity of bug reports containing crash dumps.

TABLE IV
MODELS OF BUG REPORTS CONTAINING ALARM LOGS

Model	Features	Precision	Recall
M7	Text & Categorical	70.72	72.80
M8	Text, Categorical & Alarms	70.09	72.28

Alarm logs and crash dumps are available for only 51.17% and 5.52% of all bug reports. Even when we train the models with the subset of bug reports that contain the alarms, there is no improvement in the precision and recall when compared with the model that contains only the textual and categorical features.

VI. RESULTS FOR HIGH CONFIDENCE PREDICTIONS

RQ3: What is the impact of high confidence prediction on the accuracy of triaging?

The distribution of bug reports that each team fixes is highly skewed, with a small number of teams fixing most of the bugs, see Figure 2. At Ericsson, developers suggested a novel approach by triaging only those bug reports for which the confidence in the prediction is high.

Top N Recommendation: As the problem of bug triaging deals with a large number of developers or teams, researchers are often interested to evaluate the performance of the models with top N recommendation [27] [28] [25]. In this section we report the accuracy of our model in order to be able to make comparisons with the results of other works done in this context.

We use best performing model **M3**, to evaluate the model's performance of recommending N teams. We select the top N development teams predicted by model M3 and consider a hit in accuracy if the actual team that fixed the bug is in the list of top N. In Table V we see that the model achieves 86.63% and 90.02% accuracy for recommendation of top 2 and top 3 development teams respectively. The percentage point increase in accuracy for top 2 and 3 recommendation are 7.63% and 11.02% respectively from top 1 recommendation.

TABLE V
ACCURACY OF TOPN RECOMMENDATIONS

TopN	Accuracy
Top1	79.00
Top2	86.63
Top3	**90.02**

We have 43 development teams to assign the bug reports to and achieve top1 accuracy of 79%, while other researchers tend to focus on the core developers with between 25 and 1000 developers. The previous works that have a hundreds of developers and evaluate using time split validation like us, tend to have low top 1 accuracy, for example 68% with 405 developers [27], 54% with 238 developers [25] and 43% with 77 developers. Our accuracy is also comparable with existing works that have a similar number of developers for example 77% with 76 developers and evaluated on a fixed test data [11], 28.60% with 11 developers [12].

For top 1 to top 3 recommendation, the accuracy is 79.00%, 86.63%, and 90.02% respectively.

High Confidence Bug Triaging: With TopN predictions, a developer still needs to manually assess the triage recommendation and assign it to a particular team or developer. Ericsson wants automated bug triaging and decided to automatically triage only those bug reports that the model had high confidence in the prediction (refer to III-E). Since the data is skewed with some teams fixing many bugs, the confidence that the model has in each prediction varies. There is a trade-off between accuracy and the number of predictions. We set a cutoff for the confidence score and we remove the predictions with confidence lesser than the cutoff. In Figure 6, we plot the percentage of predictions and accuracy with varying cutoff confidence scores. The percentage of predictions and accuracy are calculated using the subset of bug reports predicted with confidence higher or equal to the cutoff. The lines of prediction and accuracy intersect at confidence score around 0.6.

Table VI and Figure 6 show the impact of only triaging bug reports predicted with a confidence between 10% to 90%. When we set the confidence cutoff to 10%, we have an accuracy of 79.00% and triage 100% of the bug reports. When the cutoff confidence is 60%, We see that 83.76% of the reports are triaged with an accuracy of 85.73%. When we set

the confidence cutoff to 90%, We have an accuracy of 90.17% but the model can only triage 61.71% of the bug reports.

Since a TopN prediction will result in manual effort, Ericsson's preference is to automatically triage only those bug reports predicted with high confidence as that produces better accuracy.

> We find that when the model is 90% confident in the result, we are able to triage bug reports with a precision and recall of 89.75% and 90.17%, respectively. However, we only triage 61.71% of the total bug reports.

TABLE VI
TRIAGING BUGS WITH ABOVE A CONFIDENCE LEVEL CUTOFF. WITH A HIGHER CUTOFF, FEWER BUGS ARE TRIAGED, BUT THE ACCURACY OF THE PREDICTION IMPROVES.

Confidence Level	Triaged Bugs	Accuracy	Precision	Recall
≥ 0.1	100%	79.00	78.09	79.00
≥ 0.3	97.8%	80.19	79.36	80.19
≥ 0.5	89.18%	83.77	82.80	83.77
≥ 0.6	83.76%	85.73	84.59	85.73
≥ 0.7	78.56%	87.3	86.25	87.3
≥ 0.9	**61.71%**	**90.17**	**89.73**	**90.17**

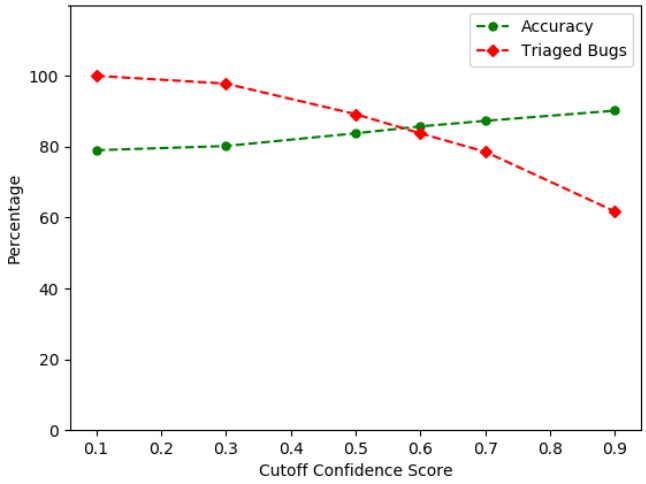

Fig. 6. The number of bugs triaged vs the prediction accuracy while varying the cutoff on confidence of the predictions

VII. Discussion of Threats to Validity

We examine bug reports of 9 large products at Ericsson. These results clearly do not generalize outside of Ericsson, however, our results are in line with previous works that examine a wide range of open source and other projects. We also examine the impact of the period used for training and the type of model.

A. Training Time Period

In the paper, we use weekly intervals to incrementally train and test the model. We train using all existing data prior to the current week of test data. Research has noted that using old data during training can reduce precision and recall [14], [27]. To address this threat, we run the best performing model M3 with the training dataset limited to two and six months respectively. At two months, we see a precision and recall of 73.67% and 75.66%. The corresponding values for six months are 77.21% and 78.73% respectively. Results of two months are quite less than using the entire training set, while experiment with six months decreases the precision and recall by less than 1 percentage point. The time period used in training is easily tuned to a company's needs.

> Removing old bug reports from training data does not improve accuracy of bug triaging at Ericsson.

TABLE VII
RESULTS WITH LIMITED TRAINING DATA

Model	Features	Time Period	Precision	Recall
M3	Text & Categorical	2 months	73.67	75.66
M3	Text & Categorical	6 months	77.21	78.73
M3	Text & Categorical	All preceding data	78.09	79.00

B. Alternative Classifiers

In this paper, we find that a logistic regression is the simplest model and has the highest precision and recall. We trained other classifiers: Naive Bayes, Linear SVM, and KNN. In Table VIII, we report the precision and recall for each classifier using the textual and categorical features of the bug reports. Naive Bayes and KNN perform poorly. While Linear SVM decreases precision and recall by only 1 percentage point than logistic regression, it requires substantially more time to train the models. We have tuned the hyper-parameters of Linear SVM and KNN. For Linear SVM we vary the regularization parameter C from 1.0 to 10.0 and get the best result at C = 2.5. For KNN, We vary the number of neighbors N up to 50. Future work could examine other models including neural networks and ensemble classifiers.

> A logistic regression classifier outperforms more sophisticated classifiers including linear SVM.

TABLE VIII
RESULTS FOR ALTERNATIVE CLASSIFIERS

Classifier	Precision	Recall
Naive Bayes	64.62	64.84
KNN	57.96	60.64
Linear SVM	77.74	78.08
Logistic Regression (M3)	**78.09**	**79.00**

VIII. Conclusion and Contributions

Bugs are inevitable in any piece of software. Manually triaging bug reports and assigning them to the right developer

or team is costly. Research into automating the bug triage process is extensive. In this paper, we examine the use of automated triaging across 9 products at Ericsson. We make three contributions.

1) We reproduce the techniques commonly used by researchers in an industrial setting. Reviewing the literature, we note that many works use cross validation or relatively small data sets to evaluate their techniques (see Table II). Cross validation is unrealistic because future bug reports are used to assign developers to past bug reports. We use a methodologically valid time split evaluation where we sequentially train and test on a large industrial data set. In our dataset, we find that older data does not reduce the precision and recall, and that 6 months of data is sufficient to preform triage.
2) Our models contain the simple textual and categorical features of bug reports as well as alarms and crash dumps. The text and categorical features outperform the more complex error information, with a precision and recall of 78% and 79% respectively. However, in our dataset only a small proportion of bug reports contain crash dumps and just over half contain alarms.
3) Although our models have a comparable accuracy to other large scale bug triaging research works, they were not sufficient for regular use at Ericsson. We can increase our accuracy to 90% when we suggest the top 3 teams, but this still requires manual triage effort to decide among the top 3. Instead, we only triage the bug reports when the model has high confidence in the prediction. Using this approach we are able to attain an accuracy of 90% on 62% of the bug reports. The manual effort is reduced by over half and high accuracy is achieved with the automatically triaged bug reports.

Automated bug triage work continues at Ericsson. The introduction of high confidence bug triaging shows promise. Furthermore, while alarms and crash logs do not improve the accuracy of triage, an effort is beginning for better storage and cleaning of crash dumps to provide a larger training set. There is hope that the tedium of triaging will become more automated in industry.

REFERENCES

[1] S. N. Ahsan, J. Ferzund, and F. Wotawa. Automatic software bug triage system (bts) based on latent semantic indexing and support vector machine. In *2009 Fourth International Conference on Software Engineering Advances*, pages 216–221. IEEE, 2009.

[2] A. Amar and P. Rigby. Mining historical test logs to predict bugs and localize faults in the test logs. In *Proceedings of the 41st International Conference on Software Engineering*. ACM, 2019.

[3] J. Anvik, L. Hiew, and G. C. Murphy. Who should fix this bug? In *Proceedings of the 28th international conference on Software engineering*, pages 361–370. ACM, 2006.

[4] J. Anvik and G. C. Murphy. Reducing the effort of bug report triage: Recommenders for development-oriented decisions. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 20(3):10, 2011.

[5] S. Banitaan and M. Alenezi. Tram: An approach for assigning bug reports using their metadata. In *2013 Third International Conference on Communications and Information Technology (ICCIT)*, pages 215–219. IEEE, 2013.

[6] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim. Duplicate bug reports considered harmfulâĂę really? In *2008 IEEE International Conference on Software Maintenance*, pages 337–345. IEEE, 2008.

[7] P. Bhattacharya, I. Neamtiu, and C. R. Shelton. Automated, highly-accurate, bug assignment using machine learning and tossing graphs. *Journal of Systems and Software*, 85(10):2275–2292, 2012.

[8] G. Canfora and L. Cerulo. Supporting change request assignment in open source development. In *Proceedings of the 2006 ACM symposium on Applied computing*, pages 1767–1772. ACM, 2006.

[9] J. Chen, X. He, Q. Lin, Y. Xu, H. Zhang, D. Hao, F. Gao, Z. Xu, Y. Dang, and D. Zhang. An empirical investigation of incident triage for online service systems. In *Proceedings of the 41st International Conference on Software Engineering*. ACM, 2019.

[10] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *Journal of machine learning research*, 9(Aug):1871–1874, 2008.

[11] A.-C. Florea, J. Anvik, and R. Andonie. Spark-based cluster implementation of a bug report assignment recommender system. In *International Conference on Artificial Intelligence and Soft Computing*, pages 31–42. Springer, 2017.

[12] H. Hu, H. Zhang, J. Xuan, and W. Sun. Effective bug triage based on historical bug-fix information. In *2014 IEEE 25th International Symposium on Software Reliability Engineering*, pages 122–132. IEEE, 2014.

[13] G. Jeong, S. Kim, and T. Zimmermann. Improving bug triage with bug tossing graphs. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 111–120. ACM, 2009.

[14] L. Jonsson, M. Borg, D. Broman, K. Sandahl, S. Eldh, and P. Runeson. Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts. *Empirical Software Engineering*, 21(4):1533–1578, 2016.

[15] L. Jonsson, D. Broman, M. Magnusson, K. Sandahl, M. Villani, and S. Eldh. Automatic localization of bugs to faulty components in large scale software systems using bayesian classification. In *2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 423–430. IEEE, 2016.

[16] H. Kagdi, M. Gethers, D. Poshyvanyk, and M. Hammad. Assigning change requests to software developers. *Journal of Software: Evolution and Process*, 24(1):3–33, 2012.

[17] S.-R. Lee, M.-J. Heo, C.-G. Lee, M. Kim, and G. Jeong. Applying deep learning based automatic bug triager to industrial projects. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pages 926–931. ACM, 2017.

[18] Z. Lin, F. Shu, Y. Yang, C. Hu, and Q. Wang. An empirical study on bug assignment automation using chinese bug data. In *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pages 451–455. IEEE, 2009.

[19] M. Linares-Vásquez, K. Hossen, H. Dang, H. Kagdi, M. Gethers, and D. Poshyvanyk. Triaging incoming change requests: Bug or commit history, or code authorship? In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 451–460. IEEE, 2012.

[20] D. Matter, A. Kuhn, and O. Nierstrasz. Assigning bug reports using a vocabulary-based expertise model of developers. In *2009 6th IEEE international working conference on mining software repositories*, pages 131–140. IEEE, 2009.

[21] H. Naguib, N. Narayan, B. Brügge, and D. Helal. Bug report assignee recommendation using activity profiles. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 22–30. IEEE Press, 2013.

[22] R. Shokripour, J. Anvik, Z. M. Kasirun, and S. Zamani. Why so complicated? simple term filtering and weighting for location-based bug report assignment recommendation. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 2–11. IEEE, 2013.

[23] R. Shokripour, Z. M. Kasirun, S. Zamani, and J. Anvik. Automatic bug assignment using information extraction methods. In *2012 International Conference on Advanced Computer Science Applications and Technologies (ACSAT)*, pages 144–149. IEEE, 2012.

[24] A. Tamrawi, T. T. Nguyen, J. M. Al-Kofahi, and T. N. Nguyen. Fuzzy set and cache-based approach for bug triaging. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pages 365–375. ACM, 2011.

[25] S. Wang, W. Zhang, and Q. Wang. Fixercache: Unsupervised caching active developers for diverse bug triage. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, page 25. ACM, 2014.

[26] W. Wu, W. Zhang, Y. Yang, and Q. Wang. Drex: Developer recommendation with k-nearest-neighbor search and expertise ranking. In *2011 18th Asia-Pacific Software Engineering Conference*, pages 389–396. IEEE, 2011.

[27] X. Xia, D. Lo, Y. Ding, J. M. Al-Kofahi, T. N. Nguyen, and X. Wang. Improving automated bug triaging with specialized topic model. *IEEE Transactions on Software Engineering*, 43(3):272–297, 2017.

[28] G. Yang, T. Zhang, and B. Lee. Towards semi-automatic bug triage and severity prediction based on topic model and multi-feature of bug reports. In *2014 IEEE 38th Annual Computer Software and Applications Conference*, pages 97–106. IEEE, 2014.

[29] T. Zhang and B. Lee. A hybrid bug triage algorithm for developer recommendation. In *Proceedings of the 28th annual ACM symposium on applied computing*, pages 1088–1094. ACM, 2013.

[30] T. Zhang, G. Yang, B. Lee, and E. K. Lua. A novel developer ranking algorithm for automatic bug triage using topic model and developer relations. In *2014 21st Asia-Pacific Software Engineering Conference*, volume 1, pages 223–230. IEEE, 2014.

[31] W. Zhang, S. Wang, Y. Yang, and Q. Wang. Heterogeneous network analysis of developer contribution in bug repositories. In *2013 International Conference on Cloud and Service Computing*, pages 98–105. IEEE, 2013.