BEE: A Tool for Structuring and Analyzing Bug Reports

Yang Song
ysong10@email.wm.edu
College of William & Mary
Williamsburg, Virginia, USA

Oscar Chaparro
oscarch@wm.edu
College of William & Mary
Williamsburg, Virginia, USA

ABSTRACT

This paper introduces Bee, a tool that automatically analyzes user-written bug reports and provides feedback to reporters and developers about the system's observed behavior (OB), expected behavior (EB), and the steps to reproduce the bug (S2R). Bee employs machine learning to (i) detect if an issue describes a bug, an enhancement, or a question; (ii) identify the structure of bug descriptions by automatically labeling the sentences that correspond to the OB, EB, or S2R; and (iii) detect when bug reports fail to provide these elements. Bee is integrated with GitHub and offers a public web API that researchers can use to investigate bug management tasks based on bug reports. We evaluated Bee's underlying models on more than 5k existing bug reports and found they can correctly detect OB, EB, and S2R sentences as well as missing information in bug reports. Bee is an open-source project that can be found at https://git.io/JfFnN. A screencast showing the full capabilities of Bee can be found at https://youtu.be/8pC48f_hClw.

CCS CONCEPTS

• **Software and its engineering → Software maintenance tools**.

KEYWORDS

Bug reporting, bug report structure, bug report quality, text analysis

ACM Reference Format:
Yang Song and Oscar Chaparro. 2020. BEE: A Tool for Structuring and Analyzing Bug Reports. In *Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '20), November 8–13, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3368089.3417928

1 INTRODUCTION

Bug reports are essential in helping developers triage, replicate, locate, and fix the bugs in the software [3, 10, 11, 21, 36, 37, 46, 47]. From the information that reporters provide in bug reports, the system's *observed (unexpected) behavior* (OB), the *steps to reproduce* the bug (S2R), and the software *expected behavior* (EB) are among the most important elements for developers [11, 15, 19, 21, 46]. These elements are typically expressed by end-users or developers in free-form natural language through issue trackers.

While these elements are essential, they are often incomplete, unclear, or not provided at all by the reporters [3, 12, 46]. Indeed, in 2016, developers from more than 1.3k open-source projects wrote a letter to GitHub expressing their frustration that bug reports are often submitted without the S2R and the system version [3]. The consequence of this is that developers often spend too much effort triaging and fixing the problems [12, 25, 46], and often, they cannot even reproduce and fix the bugs in the code [22, 45]. One of the main reasons for having low-quality bug reports is the lack of feedback and quality verification of issue trackers. In the GitHub letter [3], developers demand improvements to GitHub's issue tracker to ensure higher-quality bug reports. However, as of today, no major improvements have been made by GitHub. The alternative for some projects is to provide templates in the issues, explicitly asking for the OB, EB, S2R, and other information. Unfortunately, this approach does not guarantee that reporters will submit high-quality bug reports and developers still need to reach out to reporters asking for clarifications or more information.

In this paper, we introduce Bee (**B**ug r**E**port analyz**E**r), a tool that provides feedback to reporters and developers about the OB, EB, and S2R in bug reports. Bee is an app that extends the capabilities of GitHub's issue tracker, by analyzing incoming issues submitted by end-users on GitHub repositories. Through its machine learning models, Bee can detect if an issue reports a bug, an enhancement (*e.g.*, a feature), or a question. For bug reports, Bee can automatically identify the sentences that describe the OB, EB, and/or S2R, and detect if the reporter does not provide any of these elements.

Bee adds comments and labels to the bug report to alert reporters (and developers) about missing elements so that they can provide the information timely. Bee is meant to assist developers, by structuring the bug descriptions via automated identification and labeling of OB, EB, and S2R sentences, allowing them to quickly spot these elements. Bee is also meant to assist researchers through a public web API for OB, EB, and S2R identification, which they can use for investigating and automating tasks that are based on these elements, such as bug reproduction [31, 43], test case generation [23], bug localization [16, 17], duplicate bug report detection [18], and bug report quality assessment [15, 19].

Bee can analyze any bug report, written in any textual form and format, for any software system. Bee can be installed in seconds, in any GitHub repository. Inspired by prior work (including ours) [13, 15, 26, 33, 40, 41, 46], Bee's main vision is to perform fine-grained quality assessment of bug reports and support reporters and developers in bug reporting and management.

2 TOOL DESCRIPTION

Bee (**B**ug r**E**port analyz**E**r) is a GitHub app that analyzes incoming GitHub issues submitted by end-users, and provides feedback to reporters and developers about the system's OB, EB, and S2R.

ESEC/FSE '20, November 8–13, 2020, Virtual Event, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7043-1/20/11…$15.00
https://doi.org/10.1145/3368089.3417928

2.1 BEE's Usage Scenario and Features

Bee can be installed easily in any repository through Bee's installation website [8]. The users just have to follow a few steps for installing the app in their repositories. The current version of Bee does not require any configuration from the user.

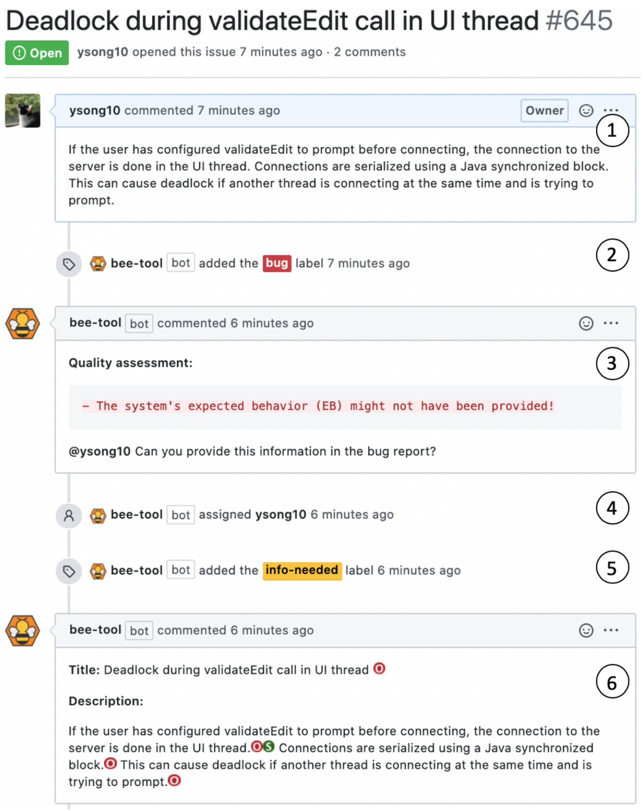
Once installed, Bee analyzes any issue reported by the project users or developers, as shown in Figure 1. Since Bee focuses on bug reports, the first step of the tool, right after an issue is submitted ①, is to automatically check if the issue describes a bug, as opposed to an enhancement (*e.g.*, a feature) or a question. If so, Bee tags the issue with the label **bug** ② and proceeds with further analysis of the bug report. Figure 1 illustrates a report submitted on GitHub that describes a bug for the Eclipse project [1]. Such a bug was originally submitted by one developer on Eclipse's issue tracker [2]. If the issue is not a bug report, Bee tags the issue with a label corresponding to the type (enhancement or question), without further analyzing its content. This initial categorization of the issue is intended to help developers prioritize and manage the reported problems.

Bee analyzes the title and description of a bug report, focusing on the OB, EB, S2R. Bee can detect when any of these elements is not provided by the reporter. In that case, Bee makes a comment in the issue ③, alerting the reporter about the missing information and asking her to provide the information. Besides, Bee assigns the issue to the reporter ④ and tags the issue with the label **info-needed** ⑤. This feedback encourages reporters to provide the information needed by the developers. If all the three elements are provided by the user, Bee makes a comment indicating the bug report appears to be complete.

Bee provides additional feedback by structuring the bug description. This feature is meant to support developers (and reporters) in understanding and assessing the quality of the OB, EB, and S2R, by helping them easily identify these elements in the bug report. The bug report is structured automatically by Bee in an additional comment ⑥, which contains the bug title and description as provided in the original issue (with the same format), but with the sentences labeled as OB Ⓞ, EB Ⓔ, or S2R Ⓢ (see Figure 1). Bee labels the sentences with the respective icon(s), at the end of the sentences, only if they convey the OB, EB, or S2R. Notice that a single sentence can convey one or more of the three types of information. The decision of labeling the sentences rather than re-organizing them into sections is made so that the (structured) bug description is easier to understand.

Bee supports any issue format, including GitHub's Markdown format, and does not impose any particular discourse on the users. This means that reporters can write their issues as they normally do. Bee treats each code snippet in the issue as a single piece of text, and identifies if they provide information about the OB, EB, and S2R. When this is the case, Bee tags the snippets at the end of the code block. Reporters get alerted about Bee's feedback via email if they have email notifications enabled on GitHub.

Finally, Bee offers a public web API for automated OB/EB/S2R identification in textual documents. Users can send API requests containing any piece of text, Bee parses the text into sentences and returns them to the user, each one marked as OB, EB, and/or S2R. These elements can be incorporated in existing or new tools, and can be leveraged to perform automated bug localization [16, 17], duplicate bug report detection [18], bug report quality assessment [15, 19], and other tasks that rely on bug reports [23, 31, 43].


Deadlock during validateEdit call in UI thread #645

Open ysong10 opened this issue 7 minutes ago · 2 comments

ysong10 commented 7 minutes ago Owner ①

If the user has configured validateEdit to prompt before connecting, the connection to the server is done in the UI thread. Connections are serialized using a Java synchronized block. This can cause deadlock if another thread is connecting at the same time and is trying to prompt.

bee-tool bot added the **bug** label 7 minutes ago ②

bee-tool bot commented 6 minutes ago

Quality assessment: ③

```
- The system's expected behavior (EB) might not have been provided!
```

@ysong10 Can you provide this information in the bug report?

bee-tool bot assigned ysong10 6 minutes ago ④

bee-tool bot added the **info-needed** label 6 minutes ago ⑤

bee-tool bot commented 6 minutes ago

Title: Deadlock during validateEdit call in UI thread Ⓞ ⑥

Description:

If the user has configured validateEdit to prompt before connecting, the connection to the server is done in the UI thread.ⓄⓈ Connections are serialized using a Java synchronized block.Ⓞ This can cause deadlock if another thread is connecting at the same time and is trying to prompt.Ⓞ

Figure 1: Bee's feedback generated for bug report #95598 from Eclipse [2], which is submitted on GitHub [1].

2.2 Under the Hood of BEE

Bee performs automated textual classification to determine the type of issue (bug, enhancement, or question) and the type of sentence (OB, EB, and/or S2R) for bug reports. Based on the sentence-level classification, Bee determines if the bug report does not contain any of the three elements.

2.2.1 *Issue Classification.*

For classifying issues, Bee relies on the classification model of Ticket Tagger [30], which is based on *fast-Text* [29]. The model is a multi-class linear neural model that receives the set of *n*-grams (*i.e.*, sequences of *n* consecutive words) extracted from the issue title and description, and outputs the probability distribution of the issue over the predefined categories [30]. The model is pre-trained using 30k issues from 12k GitHub projects and classifies an issue into one of three categories: bug report, enhancement, or question. These categories are among the default labels of GitHub Issues [4]. The model can detect bug reports, enhancements, and questions with more than 82%, 76%, and 78% precision/recall, respectively, as indicated by its evaluation [30].

2.2.2 *Sentence Classification.*

If an issue is detected as a bug report, Bee proceeds to classify each one of its sentences.

Sentence representation. The sentences of a bug report are represented as binary vectors based on *n*-grams and part-of-speech (POS) tags. This representation captures sentence vocabulary, word types, relations between consecutive words, and syntactic patterns that can help with the classification. To represent the sentences, Bee first parses the text of the bug report using the Stanford CoreNLP library [32]. The bug description is split up into sentences, considering the title as a single sentence. Then, *n*-grams and POS tags are extracted from each sentence using tokenization, lemmatization, and POS tagging. Bee extracts {1,2,3}-grams and {1,2,3}-POS tags, which correspond to sequences of one, two, and three consecutive words and POS tags, respectively. Each element of the vector represents an *n*-gram or a POS tag and takes the value one (1) if the sentence contains the element, and the value zero (0) otherwise. The size of the vector for a sentence is 902,565, which is the number of *n*-grams + POS tags found in the data we used to train the classification models. Bee keeps an index of *n*-grams and POS tags for building the vector representation of the sentences. Stop word removal is not performed as (some of) these words can help determine the meaning of the sentences (OB, EB, and S2R) [19].

Classification models. Inspired by our prior work [19], we use linear Support Vector Machines (SVMs) for classifying the sentences. SVMs are robust learning algorithms for high-dimensional and sparse data, used in text classification [28, 34]. Since the sentences in bug reports are relatively short, which means their vectors are highly sparse, SVMs are a good option for their classification.

Rather than relying on one multi-class SVM for classifying the sentences, Bee implements three binary SVMs, one for each of the information types (OB, EB, S2R). For example, the SVM for OB classifies a sentence as OB (the sentence conveys the OB) or non-OB (the sentence conveys other information than the OB). The SVMs for EB and S2R work the same way for their respective elements. By using three classifiers, Bee can detect if a sentence conveys any combination of information elements (*e.g.*, OB and EB, OB and EB and S2R, *etc.*). Also, this approach allows us to evaluate Bee's classification performance easily.

Each SVM is trained using 116,084 sentences from 5,067 bug reports, where each sentence is represented as vectors, as described above. Section 3.1 provides more details about this dataset. Since the data is imbalanced, we train the SVMs by tuning their parameter $j = C_+/C_-$, which balances the cost factors for incorrect predictions of positive ($C+$) and negative sentences ($C-$) [34]. Larger j means higher penalty on false positives (*e.g.*, non-OB sentences predicted as OB), while lower j means higher penalty on false negatives (*e.g.*, OB sentences predicted as non-OB). We select the best parameters j during the training of all three models (see Section 3.2).

Once the sentences of a bug report are represented as vectors, as described above, Bee executes each SVM model on each sentence to determine its respective information type (OB, EB, S2R, a combination of these, or other information). Based on these results, Bee can tag each sentence on GitHub.

2.2.3 Detecting Missing Elements. Since each sentence of the bug report is identified as OB, EB, S2R, or other information, Bee can use these categories to determine if the entire report fails to provide any of the three elements. If no sentence is detected as OB/EB/S2R, then it means the bug report does not provide the OB/EB/S2R. If this is the case, Bee makes a comment about this situation, alerting the user and encouraging her to provide the missing information.

2.3 Implementation

Bee is mainly implemented using Node.js runtime environment, ensuring fast, real-time processing. Bee is built as a GitHub App, which uses GitHub's Webhooks and REST API that allows integration with GitHub's issues tracker [5]. These technologies are used to obtain newly-submitted issues (their text, reporter, and other data), make comments on the issues, and assign users and labels to them. Bee's underlying classification models are implemented using the *fastText* [29] and the SVM^{light} frameworks [27], which are also known for being fast during training and execution. Bee currently analyzes a bug report in around 3-5 seconds.

3 TOOL EVALUATION

We evaluated Bee's models to measure their expected performance in identifying the OB, EB, and S2R in bug reports. Bee's website contains the replication package of the evaluation [7].

3.1 Data

We compiled the bug reports used in our prior research [16–19], which amount to 5,067 reports from 35 different software systems (*e.g.*, Eclipse, Firefox, Docker, WordPress Android, OpenJPA), spanning different domains (*e.g.*, data storage, software development, machine learning, virtualization, web browsing) and types (*e.g.*, desktop, web, mobile, libraries). The bug reports contain 116,084 sentences total (including the title), where the ones describing the OB, EB, and S2R are manually annotated to make up the ground truth. Nearly 12% of the sentences describe the OB, 2% describe the EB, and 6% describe the S2R; 82% of the sentences describe other types of information. The proportion of positive and negative instances for OB, EB, and S2R is close to 1:8, 1:54, and 1:16, respectively. This indicates the data is extremely imbalanced, which may lead to biased and less effective models. We address this issue in two ways: (1) we tune the parameter j of the SVMs; and (2) we use oversampling of the positive sentences (OB, EB, and S2R) using SMOTE [20], which generates synthetic instances nearby the positive sentences in the vector space. Although undersampling may also be helpful, it has the risk of discarding useful sentences for training the models, hence we prefer using oversampling. The average (median) # of sentences in a bug report is 23 (9), and out of these, 3 (2) are marked as OB, 1 (1) is marked as EB, and 2 (2) are marked as S2R. On average (median), 21 (6) sentences are not marked as OB, EB, or S2R.

For evaluating the detection of missing elements, we use the same data and consider a bug report missing OB, EB, and S2R as one without sentences marked as OB, EB, and S2R, respectively. Only 2% of the bug reports do not provide any OB, and nearly 69% and 45% of them do not provide any EB and any S2R, respectively.

3.2 Methodology

We performed 10-fold cross validation (10-CV) [14, 35] to measure the expected detection accuracy of each of the three SVMs models (for OB, EB, and S2R). We randomly partitioned the data into 10 equal folds, using 8 folds for training, one fold for validation, and the remaining fold for testing. At each execution of the 10-CV approach, different folds compose the data sets, thus guaranteeing that all the sentences are used for model training, validation, and testing. The validation sets as disjoint and are used for SVM tuning. The testing sets are also disjoint and used for accuracy measurement. Note that we perform 10-CV on all the sentences in our data, as opposed to the sentences of each software system. We followed this approach since our prior work revealed similar performance between project-based and cross-project evaluation settings [19]. We applied SMOTE only to the training sets, which allowed measuring the models' accuracy on actual data, without synthetic sentences.

We measured the models' detection performance using precision, recall, accuracy, and F_1 score. We tuned the parameter j of each SVM model with the values 0.1, 0.2, ..., and 1, on each validation set. For each element type (OB, EB, S2R), we train 10 SVM models (*i.e.*, for 10 j values), select the best model using the validation set, and estimate its accuracy on the testing set using the selected metrics. The best model is the one achieving the highest F_1 score. This process is repeated 10 times, following the 10-CV approach. The best SVM parameter j is 0.2 for OB and S2R, and 0.1 for EB. The overall results are computed by aggregating the true/false positives and negatives across the 10 testing sets, and then computing the metrics.

3.3 Results

The overall results of sentence classification are shown in Table 1. Bee's SVM models achieve 87%+ recall, which indicates their ability in correctly detecting OB, EB, and S2R sentences. Note that recall is most almost perfect for EB (about 98%). The results mean that Bee correctly detects the 3 OB, 1 EB, and 2 S2R sentences (out of 23 on average) expected in a typical bug report (according to our data). However, recall comes at the cost of precision, which is nearly 70% for all three models. The positive prediction rate of the models is 14.2%, 2.5%, and 7.4% for OB, EB, and S2R, respectively. This indicates that, on average, nearly 3, 1, and 2 sentences of a typical bug report are predicted as OB, EB, and S2R, however, the prediction is correct for ≈2.1, 0.7, and 1.4 sentences (on avg.), respectively.

Table 1 also shows the overall performance of Bee at detecting missing OB, EB, or S2R in an entire bug report. The results show low performance when detecting missing OB. However, only 2% of the bug reports are expected to lack this information, therefore, we anticipate a negligible effect of the misclassifications produced by the tool in practice. This observation is supported by Bee's high accuracy (≈97%). When detecting missing EB and S2R, Bee achieves substantially higher precision (93%+) and recall (70%+). Precision is almost perfect when detecting missing EB. Given the results, we can expect a few cases in which reporters are bothered with false alerts. The recall results mean that in about 1 (and 3) out of 10 bug reports, the tool fails to detect missing EB (and S2R). Since Bee is accurate in most bug reports, we expect Bee to have a positive effect on bug report quality and the bug resolution process.

In summary, Bee's models are conservative as they try not to miss any of the OB, EB, and S2R sentences in a bug report. This produces fewer false negatives, at the expense of more false positives (at sentence level). This phenomenon translates into high precision (*i.e.*, few false alarms) and lower recall (*i.e.*, more misdetections) when detecting missing elements in entire bug reports.

Table 1: Detection performance of OB, EB, and S2R sentences and missing elements in bug reports

	Sentences			Missing elements		
	OB	**EB**	**S2R**	**OB**	**EB**	**S2R**
Precision	72.6%	70.0%	72.0%	33.3%	99.7%	93.4%
Recall	87.9%	98.4%	90.8%	33.7%	88.7%	70.4%
Accuracy	94.7%	99.2%	97.4%	97.4%	92.0%	84.4%

4 RELATED WORK

A few efforts have been made to automatically identify and extract the OB, EB, and S2R from bug reports, by using heuristics and machine learning [11, 15, 19, 21, 42, 46]. Examples of heuristics include matching keywords such as "observed results" to identify the OB, or using regular expressions to detect bullets as proxies to the S2R [11, 21, 46]. Since these approaches often fail to capture the diverse discourse [19] found in bug reports, machine-learning-based approaches, like the ones Bee implements, have been proposed [15, 19, 42]. Our prior work [19] used SVMs based on textual features to detect when bug reports lack the EB and S2R. More recently, SVM- and sequence-labeling-based techniques have been proposed to identify S2R sentences in bug reports from mobile apps [15, 42].

Many approaches have been proposed to classify issues into bug reports, feature requests, enhancements, questions, and other categories [6, 9, 24, 25, 30, 38, 39, 44]. These approaches typically implement machine learning models that use textual features for classification. Bee uses Ticket Tacker's pre-trained model [30] to identify if a newly-submitted issue reports a bug.

Different from prior work, Bee identifies the OB, EB, and S2R, at sentence level, in bug reports written by end-users in any form and for any software system. Bee's features enable many applications in bug management, as indicated by prior work [15–19, 23, 31, 43].

5 CONCLUSIONS AND FUTURE WORK

Bee is an open-source tool, integrated with GitHub, that uses machine learning to automatically (1) detect the type of user-written GitHub issues (bug report, enhancement, or question), (2) identify and label sentences describing the system's observed behavior (OB), expected behavior (OB), the steps to reproduce (S2R) the bug in bug reports, and (3) detect if these elements are not provided by the reporters. Bee is meant to alert reporters about missing information in their bug reports, assist developers on bug triage and resolution, and foster new research developments on automated bug management. The evaluation of Bee's underlying models, using 5k+ bug reports, provides evidence of its high accuracy in identifying the OB, EB, and S2R in bug reports. Given the results, we anticipate Bee can have a positive effect on bug report quality and bug management, yet this is to be confirmed by our planned user studies. Improvements to Bee include the implementation of a mechanism to automatically retrain Bee's models based on user feedback and autocompleting missing bug report elements.

ACKNOWLEDGMENTS

This work was partially supported by the NSF grant CCF-1955853.

REFERENCES

[1] 2005. Bug report #95598 from Eclipse submitted on GitHub. https://github.com/ysong10/fast_ALS/issues/645.

[2] 2005. Original bug report #95598 from Eclipse. https://bugs.eclipse.org/bugs/show_bug.cgi?id=95598.

[3] 2016. An open letter to GitHub from the maintainers of open source projects. Available online: https://github.com/dear-github/dear-github.

[4] 2020. Default labels on GitHub Issues. https://help.github.com/en/github/managing-your-work-on-github/about-labels.

[5] 2020. GitHub Developer: Using the GitHub API in your app. https://developer.github.com/apps/quickstart-guides/using-the-github-api-in-your-app/.

[6] 2020. Issue-Label bot. https://github.com/marketplace/issue-label-bot.

[7] 2020. Replication package of Bee's evaluation. https://github.com/sea-lab-wm/bee-tool.

[8] 2020. Bee's installation website. https://github.com/apps/bee-tool.

[9] Giuliano Antoniol, Kamel Ayari, Massimiliano Di Penta, Foutse Khomh, and Yann-Gaël Guéhéneuc. 2008. Is It a Bug or an Enhancement?: A Text-based Approach to Classify Change Requests. In *Proceedings of the Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds*. 304–318.

[10] Nicolas Bettenburg, Rahul Premraj, Thomas Zimmermann, and Sunghun Kim. 2008. Duplicate Bug Reports Considered Harmful ... Really?. In *Proceedings of the International Conference on Software Maintenance (ICSM'08)*. 337–345.

[11] Nicolas Bettenburg, Rahul Premraj, Thomas Zimmermann, and Sunghun Kim. 2008. Extracting Structural Information from Bug Reports. In *Proceedings of the International Working Conference on Mining Software Repositories (WCRE'08)*. 27–30.

[12] Silvia Breu, Rahul Premraj, Jonathan Sillito, and Thomas Zimmermann. 2010. Information Needs in Bug Reports: Improving Cooperation Between Developers and Users. In *Proceedings of the Conference on Computer Supported Cooperative Work (CSCW'10)*. 301–310.

[13] Gemma Catolino, Fabio Palomba, Andy Zaidman, and Filomena Ferrucci. 2019. Not all bugs are the same: Understanding, characterizing, and classifying bug types. *Journal of Systems and Software* 152 (2019), 165–181.

[14] Gavin C Cawley and Nicola LC Talbot. 2010. On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research* 11, Jul (2010), 2079–2107.

[15] Oscar Chaparro, Carlos Bernal-Cárdenas, Jing Lu, Kevin Moran, Andrian Marcus, Massimiliano Di Penta, Denys Poshyvanyk, and Vincent Ng. 2019. Assessing the quality of the steps to reproduce in bug reports. In *Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE'19)*. 86–96.

[16] Oscar Chaparro, Juan Manuel Florez, and Andrian Marcus. 2017. Using observed behavior to reformulate queries during text retrieval-based bug localization. In *Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME'17)*. IEEE, 376–387.

[17] Oscar Chaparro, Juan Manuel Florez, and Andrian Marcus. 2019. Using bug descriptions to reformulate queries during text-retrieval-based bug localization. *Empirical Software Engineering* 24, 5 (2019), 2947–3007.

[18] Oscar Chaparro, Juan Manuel Florez, Unnati Singh, and Andrian Marcus. 2019. Reformulating queries for duplicate bug report detection. In *Proceedings of the IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER'19)*. IEEE, 218–229.

[19] Oscar Chaparro, Jing Lu, Fiorella Zampetti, Laura Moreno, Massimiliano Di Penta, Andrian Marcus, Gabriele Bavota, and Vincent Ng. 2017. Detecting Missing Information in Bug Descriptions. In *Proceedings of the 11th ACM Joint Meeting on the Foundations of Software Engineering (ESEC/FSE'17)*. 396–407.

[20] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research* 16 (2002), 321–357.

[21] Steven Davies and Marc Roper. 2014. What's in a Bug Report?. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM'14)*. 26:1–26:10.

[22] Mona Erfani Joorabchi, Mehdi Mirzaaghaei, and Ali Mesbah. 2014. Works for Me! Characterizing Non-reproducible Bug Reports. In *Proceedings of the Working Conference on Mining Software Repositories (MSR'14)*. 62–71.

[23] Mattia Fazzini, Martin Prammer, Marcelo d'Amorim, and Alessandro Orso. 2018. Automatically translating bug reports into test cases for mobile apps. In *Proceedings of the 27th ACM International Symposium on Software Testing and Analysis (ISSTA'18*. 141–152.

[24] Don Goodman-Wilson. 2018. Automating issue triage with GitHub and Recast.AI. https://github.blog/2018-10-31-automating-issue-triage-with-github-and-recastai/.

[25] Philip J. Guo, Thomas Zimmermann, Nachiappan Nagappan, and Brendan Murphy. 2010. Characterizing and Predicting Which Bugs Get Fixed: An Empirical Study of Microsoft Windows. In *Proceedings of the International Conference on Software Engineering (ICSE'10)*. 495–504.

[26] Pieter Hooimeijer and Westley Weimer. 2007. Modeling Bug Report Quality. In *Proceedings of the 22nd International Conference on Automated Software Engineering (ASE'07)*. 34–43.

[27] Thorsten Joachims. 1998. *Making Large-Scale SVM Learning Practical*. LS8-Report 24. Universität Dortmund, LS VIII-Report.

[28] Thorsten Joachims. 1998. Text categorization with support vector machines: Learning with many relevant features. In *Proceedings of the European Conference on Machine Learning*. Springer, 137–142.

[29] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2016. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759* (2016).

[30] Rafael Kallis, Andrea Di Sorbo, Gerardo Canfora, and Sebastiano Panichella. 2019. Ticket Tagger: Machine Learning Driven Issue Classification. In *Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME'19)*. 406–409.

[31] Gün Karagöz and Hasan Sözer. 2017. Reproducing failures based on semiformal failure scenario descriptions. *Software Quality Journal* 25, 1 (2017), 111–129.

[32] Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL'14)*. 55–60.

[33] Kevin Moran, Mario Linares-Vásquez, Carlos Bernal-Cárdenas, and Denys Poshyvanyk. 2015. Auto-completing Bug Reports for Android Applications. In *Proceedings of the Joint Meeting on Foundations of Software Engineering (FSE'15)*. 673–686.

[34] K. Morik, P. Brockhausen, and T. Joachims. 1999. Combining Statistical Learning with a Knowledge-based Approach – a Case Study in Intensive Care Monitoring. In *International Conference on Machine Learning (ICML)*. Bled, Slowenien, 268–277.

[35] Sebastian Raschka. 2018. Model evaluation, model selection, and algorithm selection in machine learning. *arXiv preprint arXiv:1811.12808* (2018).

[36] Swarup Kumar Sahoo, John Criswell, and Vikram Adve. 2010. An Empirical Study of Reported Bugs in Server Software with Implications for Automated Bug Diagnosis. In *Proceedings of the International Conference on Software Engineering (ICSE'10)*. 485–494.

[37] Tommaso Dal Sasso, Andrea Mocci, and Michele Lanza. 2016. What Makes a Satisficing Bug Report?. In *Proceedings of the International Conference on Software Quality, Reliability and Security (QRS'16)*. 164–174.

[38] Ferdian Thung, David Lo, and Lingxiao Jiang. 2012. Automatic defect categorization. In *Proceedings of the 19th Working Conference on Reverse Engineering (WCRE'12)*. IEEE, 205–214.

[39] Bowen Xu, David Lo, Xin Xia, Ashish Sureka, and Shanping Li. 2015. EFSPredictor: Predicting Configuration Bugs with Ensemble Feature Selection. In *Proceedings of the Asia-Pacific Software Engineering Conference (ASPEC'15)*. 206–213.

[40] Marcelo Serrano Zanetti, Ingo Scholtes, Claudio Juan Tessone, and Frank Schweitzer. 2013. Categorizing Bugs with Social Networks: A Case Study on Four Open Source Software Communities. In *Proceedings of the International Conference on Software Engineering (ICSE'13)*. 1032–1041.

[41] Tao Zhang, Jiachi Chen, He Jiang, Xiapu Luo, and Xin Xia. 2017. Bug Report Enrichment with Application of Automated Fixer Recommendation. In *Proceedings of the 25th International Conference on Program Comprehension (ICPC'17)*. 230–240.

[42] Yu Zhao, Kye Miller, Tingting Yu, Wei Zheng, and Minchao Pu. 2019. Automatically Extracting Bug Reproducing Steps from Android Bug Reports. In *International Conference on Software and Systems Reuse (ICSR'19)*. Springer, 100–111.

[43] Yu Zhao, Tingting Yu, Ting Su, Yang Liu, Wei Zheng, Jingzhi Zhang, and William GJ Halfond. 2019. Recdroid: automatically reproducing android application crashes from bug reports. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 128–139.

[44] Yu Zhou, Yanxiang Tong, Ruihang Gu, and Harald Gall. 2016. Combining text mining and data mining for bug report classification. *Journal of Software: Evolution and Process* 28, 3 (2016), 150–176.

[45] Thomas Zimmermann, Nachiappan Nagappan, Philip J. Guo, and Brendan Murphy. 2012. Characterizing and Predicting Which Bugs Get Reopened. In *Proceedings of the International Conference on Software Engineering (ICSE'12)*. 1074–1083.

[46] Thomas Zimmermann, Rahul Premraj, Nicolas Bettenburg, Sascha Just, Adrian Schröter, and Cathrin Weiss. 2010. What Makes a Good Bug Report? *IEEE Transactions on Software Engineering* 36, 5 (2010), 618–643.

[47] Thomas Zimmermann, Rahul Premraj, Jonathan Sillito, and Silvia Breu. 2009. Improving Bug Tracking Systems. In *Proceedings of the International Conference on Software Engineering (ICSE'09)*. 247–250.