frontiers | Frontiers in Computer Science

TYPE Original Research
PUBLISHED 22 June 2023
DOI 10.3389/fcomp.2023.1032440

OPEN ACCESS

EDITED BY
Yingxu Wang,
University of Calgary, Canada

REVIEWED BY
Rubing Huang,
Macau University of Science and Technology, Macao SAR, China
Anjali Goyal,
Sharda University, India

*CORRESPONDENCE
Yuki Noyori
✉ akskw-luck@akane.waseda.jp

RECEIVED 30 August 2022
ACCEPTED 09 May 2023
PUBLISHED 22 June 2023

CITATION
Noyori Y, Washizaki H, Fukazawa Y, Ooshima K, Kanuka H and Nojiri S (2023) Deep learning and gradient-based extraction of bug report features related to bug fixing time. *Front. Comput. Sci.* 5:1032440. doi: 10.3389/fcomp.2023.1032440

Deep learning and gradient-based extraction of bug report features related to bug fixing time

Yuki Noyori[1,2]*, Hironori Washizaki[1], Yoshiaki Fukazawa[1], Keishi Ooshima[2], Hideyuki Kanuka[2] and Shuhei Nojiri[2]

[1]Department of Computer Science and Engineering, Waseda University, Tokyo, Japan, [2]Research & Development Group, Hitachi, Ltd., Tokyo, Japan

Bug reports typically contain detailed descriptions of failures, hints at the location of the corresponding defects, and discussions. Developers usually resolve bugs using comments in descriptions and discussions. The time to fix a bug varies greatly. Previous studies have investigated bug reports, but the influence of comments on bug fixing time is not well understood. This study adopts a convolutional neural network (CNN) and gradient-based visualization approach called Grad-cam to elucidate the impact of comments on bug fixing time and extract features. A feature represents an observed characteristic in a bug report when processing via deep learning. Specifically, CNN classifies bug reports, and then Grad-cam visualizes the decision basis of CNN by identifying the top 10 word sequences used in the prediction. Here, the features are major word sequences extracted by Grad-cam. In an experiment, the proposed method classified more than 36,000 actual bug reports from Bugzilla with an accuracy of 75%–80%. Additionally, the visualization highlighted differences in the stack trace and word abstraction by bug fixing time. Bug reports with short bug fixing times are concrete, whereas those with a long bug fixing time are abstract.

KEYWORDS

deep learning, OSS, bug report, Grad-cam, feature extraction

1. Introduction

Bug reports provide details of glitches such as code defects in software. The content quality varies widely (Zimmermann et al., 2010), and there is not a universal format. Some reports hint at the location or root cause of the defect (i.e., bug), while others provide scant information.

Bug reports serve different purposes. Software testers use bug reports to record failures and bugs, whereas developers reference them to fix bugs. Additionally, Open Source Software (OSS) projects collect bug reports in repositories to accumulate knowledge.

Even if a bug report is described, it does not guarantee that the corresponding issue will be resolved. Factors influencing bug fixing time are the attributes of the initial report and the presence of comments (Panjer, 2007; Zhang et al., 2012). Another factor may be the description and the discussion in the bug report, but their influences are not well researched. Previous studies have evaluated efficient bug fixing using bug reports. Some studies have aimed to predict, assign, and search bug reports (Shokripour et al., 2012; Youm et al., 2015; Han et al., 2017; Noyori et al., 2018). One study investigated bug localization. Recently, deep learning has been applied to bug-related activities, including identifying, predicting, and tracing bug reports (Guo et al., 2017; Li et al., 2017; Palacio et al., 2019). However, the basis for many deep learning predictions remains a black box.

This study examines the influence of descriptions and discussions on bug reports. Here, the comments represent the description and discussion. The objective of this study is to extract features automatically to support bug reporters and software developers having better understanding and making better descriptions and discussions in bug reports from the viewpoint of bug fixing.

To achieve the objective, this study proposes a bug report fixing time prediction and visualization method, as shown in Figure 2, with correspondences to the following research questions.[1] First, bug reports are classified into two groups (long and short) based on fixing time as determined by a convolutional neural network (CNN). CNN is a common deep learning technique to predict binary classifications. In this study, the explanatory variable is the fixing time. Then, Gradient-weighted Class Activation Mapping (Grad-cam) (Selvaraju et al., 2017) can visualize the decision basis prediction by CNN as it reveals the word sequence used in the prediction. Examining the word sequences characterizes bug report comments by fixing time and reveals differences in the stack traces and the discussion abstraction level.

This study has two main contributions:

- The CNN-based bug report fixing time prediction method is proposed to classify bug reports. CNN learns the binary classification of the bug fixing time (long or short).
- The proposed method is also capable of visualizing the decision basis prediction by CNN to identify features that represent an observed characteristic in a bug report when processing via deep learning. Here, the features are word sequences extracted by Grad-cam. Specifically, word sequences are divided into parts of speech, and the distribution is assessed to determine the decision basis.
- The proposed prediction and visualization system is evaluated for more than 36,000 actual bug reports from Bugzilla.[2] This study confirmed that our proposed method accurately relates comments in bug reports with fixing time lengths. Furthermore, this study confirmed that our method can identify the differences in bug report features by fixing time categories.

The rest of this study is structured as follows. Section 2 discusses related studies. Section 3 describes our proposed method. Section 4 presents our experiment. Section 5 identifies threats to validity. Section 6 lists recommendations to improve bug reports. Finally, Section 7 provides the conclusion and future directions.

2. Related work

Previous studies have analyzed bug reports using deep learning. The similarities and differences of our proposed method to existing studies are presented below.

2.1. Predicting bug fixing times and severity

The task of bug fixing time prediction can be categorized as a text classification task. In modern machine learning, a typical model of text classification is structured as follows: a set of training text samples that are already labeled with a class is given; then, the knowledge related to the target class is either manually defined by human experts or automatically extracted by computer programs; the knowledge is, then, used to classify new data samples (Kowsari et al., 2019).

Classical machine learning classification algorithms such as Naive Bayes, Support Vector Machine (SVM), Hidden Markov Model (HMM), and Random Forest remain helpful under certain situations. However, deep learning-based classifiers have shown better efficiency and effectiveness than most of the classical machine learning approaches in the background of big data. There are many deep learning models for text classification available today, including RNN-based models, CNN-based models, capsule neural networks, models with attention mechanisms, and memory-augmented networks (Minaee et al., 2022).

Both machine learning and deep learning have been employed to predict bug fixing times (Giger et al., 2010; Marks et al., 2011; Zhang et al., 2013; Habayeb et al., 2018; Lee et al., 2020; Gomes et al., 2022, 2023). A summary of studies that conducted mining on bug management databases to achieve learning models related to bug fixing times is presented in Table 1, which is an extension of the research summarization by (Lee et al., 2020), with additions of the latest studies and our method.

Giger et al. (2010) adopted Decision Tree analysis to predict and bin bug reports into two classes by utilizing bug report attributes: fast and slow. Marks et al. (2011) used a decision tree-based algorithm, particularly Random Forest, to classify a bug given the bug report attributes into one of the three classes: fixed in less than 3 months, 1 year, and 3 years. Zhang et al. (2013) utilized k-Nearest Neighbor (kNN) to predict for a given bug report whether the fix will be fast (i.e., short time) or slow (i.e., long time) by utilizing bug report attributes including the severity and priority, which inspired us to use machine learning methods for predicting bug fixing time. Habayeb et al. (2018) proposed an approach using Hidden Markov Models (HMMs) and temporal sequences of developer activities to identify bug reports with expected bug fixing times. Lee et al. (2020) proposed an approach for predicting bug fixing times over time by adopting deep neural networks including Residual Long Short-Term Memory (RLSTM) and Bi-direction Long Short-Term Memory (BLSTM) models for analyzing log streams of bug-related activities and text data retrieved from bug tracking systems. Gomes et al. (2023) compared various well-known ML classifiers such as Random Forest and Support Vector Machine (SVM) on long-lived bug prediction using Bidirectional Encoder Representations from Transformers (BERT) and Term Frequency–Inverse Document Frequency (TF-IDF)-based feature extraction. Furthermore, one study used CNN to predict whether an issue is security-related (Palacio et al., 2019), while another predicted the severity of software vulnerability with CNN (Han et al., 2017). As shown in the table, although it is hard to compare them directly due to the difference in target datasets, our method can be competitive in terms of prediction performance with a particular focus on bug report text.

1 This study substantially extends our preliminary 6-page conference paper presented at ICAICA 2021 (Noyori et al., 2021b). Explanations of the proposed method and related works are considerably revised and expanded in a well-structured paper format.

2 Bugzilla is a trademark of the Mozilla Foundation. https://bugzilla.mozilla.org/.

TABLE 1 Summary of research on bug fixing time adopted by Lee et al. (2020) with additions of the latest studies and our method.

Paper	Problem definition	Input data	Learning model	Metric (%)
Giger et al. (2010)	Bug fixing time binary classification	Report attributes	Decision tree	F1 score (66–68)
Marks et al. (2011)	Bug fixing time 3-class classification	Report attributes	Random Forest	Accuracy (64–67)
Zhang et al. (2013)	Bug fixing time binary classification	Report attributes	kNN	F1 score (68–77)
Habayeb et al. (2018)	Bug fixing time binary classification	Developer activity features	HMM	F1 score (74), Accuracy (71)
Lee et al. (2020)	Bug fixing time multi-class (from 2-class to 9-class) classification	Bug-related activities	RLSTM and BLSTM	Accuracy (66–74)
Gomes et al. (2023)	Bug fixing time binary classification	Report text	SVM (and others) with BERT and TF-IDF	Accuracy (57–62)
Our method	Bug fixing time binary classification	Report text	CNN with word embedding	Accuracy (around 75)

Furthermore, to the best of our knowledge, important words and characteristics to predict bug fixing time have yet to be determined. Although there has been a comparison of different text feature extraction approaches in terms of the accuracy of machine learning classifiers (Gomes et al., 2022, 2023), important concrete features have yet to be examined well. The influence of comments on bug fixing time is not well understood.

Unlike previous reports, this study not only employs CNN but also analyzes the decision basis for the CNN prediction. The novelty of the proposed approach lies in the visualization of the decision basis prediction by CNN to identify features that represent an observed characteristic in a bug report when processing.

In future, we plan to compare our method directly with approaches based on other existing state-of-the-art models (such as BERT), to confirm the model performance by additional comparative experiments.

2.2. Classifying bug reports and detecting duplicates

Machine learning and deep learning have been used to classify bug reports and detect duplicates. Previous research indicates that these approaches are highly accurate (He et al., 2020; Neysiani and Morteza, 2020; Ahmed et al., 2021; Isotani et al., 2021) and can classify requirement documents (Hey et al., 2020). However, previous studies have yet to fully visualize the classification and duplicate detection processes. To address this shortcoming, this study extracts important words related to bug fixing time by visualizing the decision basis of deep learning-based predictions.

2.3. Evaluating bug reports

Bug report contents have been investigated. Using a questionnaire, one study identified differences between the information shared by reporters and useful information according to developers (Yusop et al., 2016). For example, reporters indicated that titles and summaries are important, whereas developers felt that they are unnecessary. Developers want to know the cause of the issue, but this information is often omitted. A different study investigated supplementing omitted content using information from previous bug reports (Zhang et al., 2017). Another study identified typical patterns in the discussion of bug reports (Noyori et al., 2021a). Unlike these previous studies, which only evaluated the presence or lack of content, this study implements deep learning and visualization to investigate specific descriptions such as important words in relation to bug fixing times.

2.4. Visualizing self-attention

Few studies have investigated deep learning applications related to the visualization of bug reporting. One study categorized defect reports using BERT and denoted essential words in a categorization by visualizing self-attention (Hirakawa et al., 2020). Although our method also visualizes important regions using deep learning, this study analyzes the application results with an emphasis on the bug fixing time. Moreover, the deep learning model, in this study, differs from the previous study. In future, these two models should be compared.

2.5. Determining project-specific terms

Previous studies have applied natural language processing to identify words specific to a target project or requirement. One study extracted both a general document corpus and words specific to the target requirement by focusing on the main words constituting compound nouns (Gacitua et al., 2010). Unlike the previous study, this study applies deep learning to identify important words related to bug fixing time. In future, a study should investigate a combination of natural language processing and deep learning.

3. Proposed method: deep learning-based prediction and visualization

In this section, we first show the motivation of predicting bug report fixing time prediction and visualization. Based on the motivation, we present the entire process of our prediction and visualization method, followed by technical details of each step.

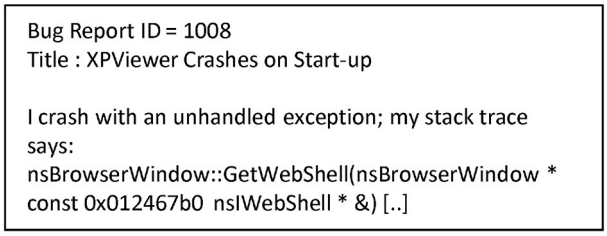

FIGURE 1
Motivating example of a bug report.

3.1. Motivation

The time to fix a bug varies greatly. Previous studies have investigated bug reports, but the influence of comments on bug fixing time is not well understood. Figure 1 shows a motivating example of a bug report with a short fixing time taken from Bugzilla, a bug tracking system. This example contains a description of the encountered issue stating *"I crash with an unhandled exception; my stack trace says: [...]."* Here, the description seems to contain general and likely less informative word sequences such as "I crash with [...]" and somewhat specific, probably more informative sequences such as "stack trace says [...]" from the viewpoint of usefulness for bug fixing.

Such influence of comments on bug fixing time should be understood well to support bug reporters and software developers having better understanding and making better descriptions and discussions in bug reports from the viewpoint of bug fixing.

3.2. Overview of the proposed method

To achieve the objective, this study proposes a bug report fixing time prediction and visualization method, as shown in Figure 2 with correspondences to the following research questions.[3] First, bug reports are classified into two groups (long and short), based on fixing time as determined by a convolutional neural network (CNN). CNN is a common deep learning technique to predict binary classifications. In this study, the explanatory variable is the fixing time. Then, Gradient-weighted Class Activation Mapping (Grad-cam) (Selvaraju et al., 2017) can visualize the decision basis prediction by CNN as it reveals the word sequence used in the prediction. Examining the word sequences characterizes bug report comments by fixing time and reveals differences in the stack traces and the discussion abstraction level.

Here, important words in bug report comments are extracted using CNN. Our method employs two steps as follows: prediction and visualization. In step 1, CNN learns the binary classification of the bug fixing time (long or short). In step 2, Grad-cam compares the extracted important words.

3 This study substantially extends our preliminary 6-page conference paper presented at ICAICA 2021 (Noyori et al., 2021b). Explanations of the proposed method and related studies are considerably revised and expanded in a well-structured paper format.

3.3. Predicting bug fixing time

Figure 3 shows the prediction process. Deep learning divides the bug fixing times into binary values of short and long. The first step to predicting bug fixing time is to divide the bug reports into two groups according to their actual bug fixing times. Next, the CNN model is trained using the first half of the bug report comments to learn the classification by group. The training uses only the first half of the report because predicting bug fixing time early in the bug fixing process is more practical. Finally, the trained CNN model classifies new bug reports by fixing time.

CNN was originally designed for computer vision tasks such as entity recognition (Lawrence et al., 1997). Later, CNN has been adopted for text classification such as Dynamic CNN (DCNN) (Kalchbrenner et al., 2014), and a simpler CNN-based model which applies only one convolutional layer (Zhang and Wallace, 2017). In recent years, more CNN-based models have proven to perform well in natural language processing tasks such as text classification. As former studies show, CNN models could achieve the state-of-the-art result on bug report analysis tasks. That is why we use the CNN network as the fundamental building block of our approach.

Figure 4 shows the structure of the CNN model that we employed. This model comprises multiple layers with the input and output as follows.

1. Input: The model accepts the text of a bug report and tokenizes it into a set of words.
2. Embedding layer: The embedding layer converts words into 64-dimensional vector representations.
3. Convolution layer: The convolution layer applies a filter with its kernel size 3 to the given vectors to create outputs that summarize the input.
4. Max-pooling layer: The pooling layer downs the sampling inputs by taking the maximum value from each pool. In our setting, the layer makes the input size in half.
5. Flattening and fully-connected layers: The flatten layer makes the multidimensional input one-dimensional, followed by fully-connected layers (i.e., dense layers) with the ReLU activation and dropout.
6. Output: Finally, the Softmax function is applied to calculate the probabilities of short and long fixing times.

3.4. Visualizing the decision basis

Figure 5 shows the visualization process. Our method first applies Grad-cam to the CNN model to obtain and visualize word sequences, which are important for prediction. Then, our method divides word sequences into parts of speech, and the distribution is assessed to determine the decision basis. Finally, the differences between the words and parts of speech are compared by fixing time categories.

As machine learning methods have become more complex in architecture and more prevalent in scientific research and industrial practices, the need to explain and interpret machine learning models has increased substantially. Grad-cam is a well-accepted gradient-based visualization approach to determine the basis of the decision from the created CNN-based model. Specifically,

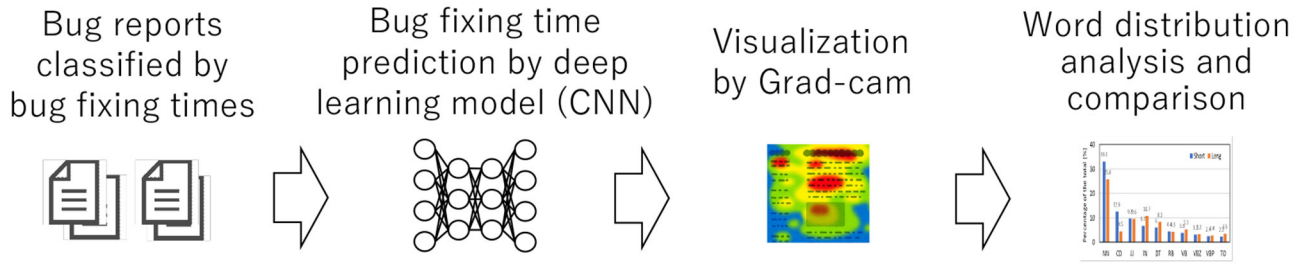

FIGURE 2
Overview of the proposed method.

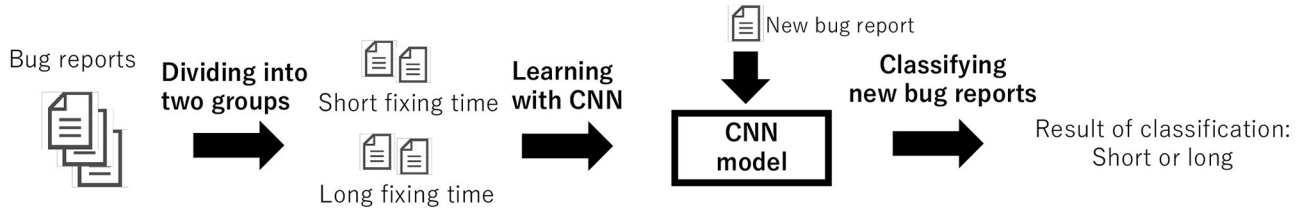

FIGURE 3
Step 1 of the proposed method: prediction.

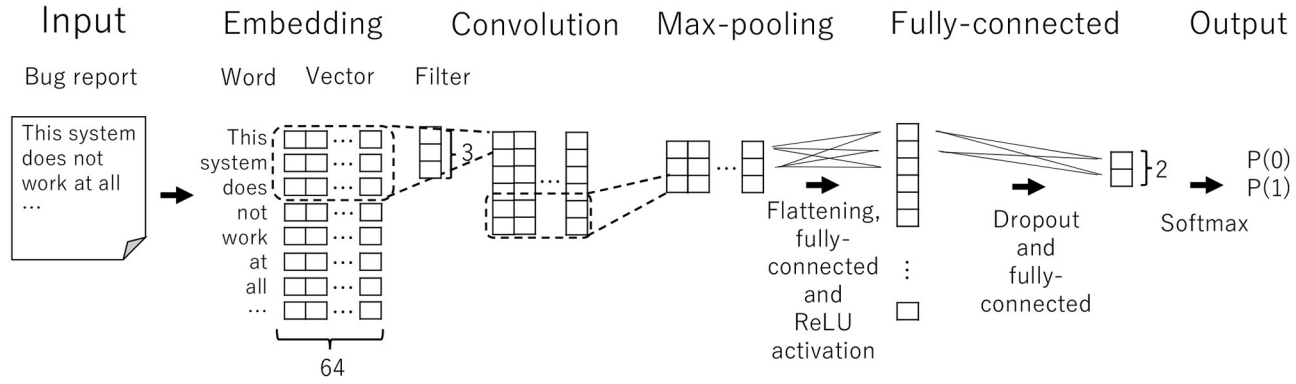

FIGURE 4
Adopted CNN architecture.

Grad-cam uses the gradient information flowing into the last convolutional layer of the CNN model to assign important values to each neuron for a particular decision of interest (Selvaraju et al., 2017).

In our model, the gradient of the convolution layer can visualize the attention a word receives. Here, the gradient at the end of the convolution layer of the CNN model is used to extract the basis of a decision. This study assumes that the top 10 words are important word sequences for prediction. Then, the word sequences and parts of speech are extracted. Finally, the features in bug reports with short bug fixing times are compared with those with long fixing times.

Figure 6 shows a visualization result of the motivating bug report, which is the same as that shown in Figure 6 with additional highlights. Red indicates an important word sequence used as a decision basis. In this example, three strings are red because the kernel is 3-gram. Specifically, the sequence in terms of the stack trace and concrete terms and values such as "const" are extracted. As highlighted by this example, the description of the stack trace and concrete terms are features of bug reports with short fixing time.

4. Experimental evaluation

This study aims to answer three research questions (RQs) as follows:

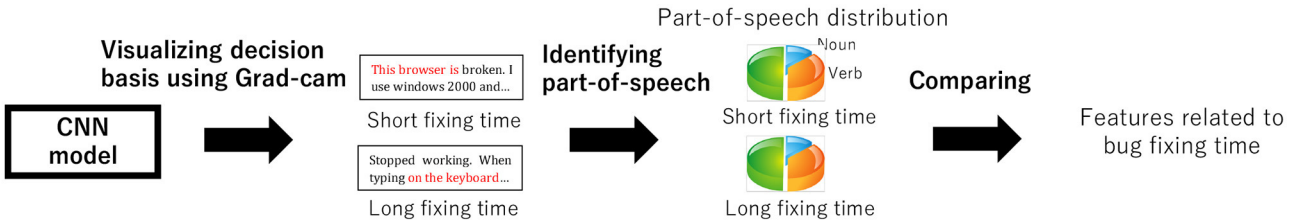

FIGURE 5
Step 2 of the proposed method: visualization.

Bug Report ID = 1008
Title : XPViewer Crashes on Start-up

I crash with an unhandled exception; my stack trace says:
nsBrowserWindow::GetWebShell(nsBrowserWindow * const 0x012467b0 nsIWebShell * &) [..]

FIGURE 6
Visualization of a bug report.

RQ1. **Does our CNN-based prediction method precisely classify bug reports by fixing time?** This RQ assesses whether our CNN-based prediction method can classify bug reports. To evaluate whether our proposed method accurately relates comments in bug reports with fixing time lengths, the accuracy of binary classification is determined for more than 36,000 actual bug reports from Bugzilla.

RQ2. **What words compose the basis for CNN's decisions in bug reports?** The features are word sequences extracted by Grad-cam. Specifically, word sequences are divided into parts of speech, and the distribution is assessed to determine the decision basis.

RQ3. **Does bug fixing time influence the basis of CNN's decision?** To identify the differences in bug report features by fixing time, the differences between the words and parts of speech are compared by fixing time categories.

The three RQs were evaluated experimentally to clarify the features of bug reports using CNN.

4.1. Dataset

We constructed and used a dataset of actual bug reports taken from multiple OSS products. Bugzilla@Mozilla[4] deals with Mozilla-related products and is available to the public. This study employed unique bug reports with IDs of 1,000 to 50,000 in Bugzilla. The target dataset includes a range of Mozilla-related products, including SeaMonkey[5] and Firefox[6]. Among those 49,001 bug reports from Bugzilla, only 36,274 reports were used in the experiment because unresolved bug reports were excluded.

In future, we plan to validate the universality of our system by evaluating the performance using other real-world datasets with more reports.

4.2. RQ1: Predicting bug fixing time using CNN

The threshold for a short or long bug fixing time should be between the average and the median. Here, the average and the median were 188 and 17 days, respectively. For convenience, the threshold between a short and long bug fixing time was set to 100 days. Hence, the binary classification indicated that 27,350 reports had short bug fixing times and 8,924 reports had long ones.

The dataset was randomly divided into training and test data. CNN used 32,646 reports (90%) as training data, and the remaining 3,628 (10%) as test data. Figure 7 shows the accuracy and loss function values. The results in the training data are denoted by acc and loss, while the results in the test data are represented by val acc and val loss. The x-axis is the epoch number, and the y-axis indicates the value of accuracy and loss. For learning with an epoch of 4 or 5, the accuracy in the training data exceeded 99%, with a validation accuracy between 75 and 80%.

In future, we plan to conduct more experiments to implement the N-fold cross-validation to avoid overfitting by dividing all reports into N groups using various N values (such as two-fold and 10-fold) and calculating the average of the results.

RQ1. Does our CNN-based prediction method precisely classify bug reports by fixing time? **Our prediction accuracy based on CNN is 75–80%.**

4.3. RQ2: Words used as a decision basis

According to Grad-cam, bug reports with short fixing times contained word sequences about concrete items (Figure 8). For example, the visualization highlighted the software version and the

4 Mozilla is a trademark of the Mozilla Foundation.

5 SeaMonkey is a trademark of the Mozilla Foundation.

6 Firefox is a trademark of the Mozilla Foundation.

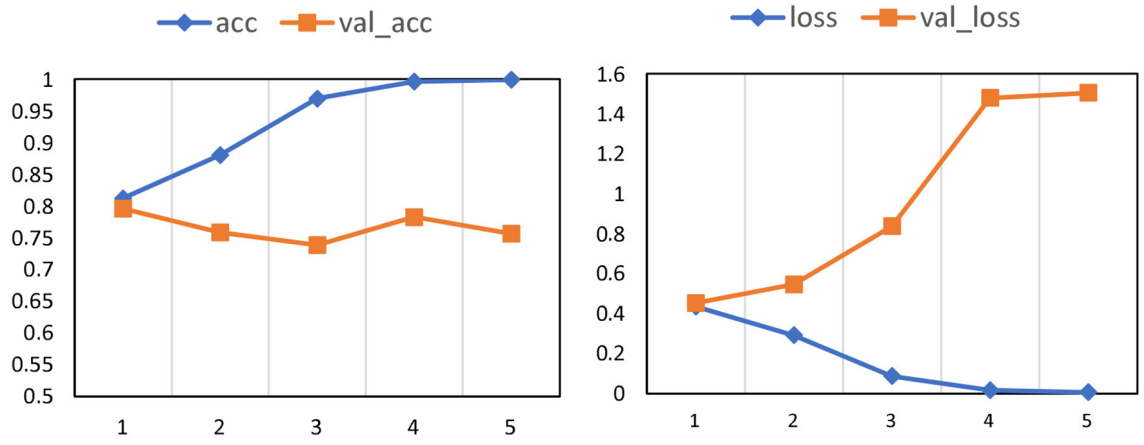

FIGURE 7
Learning results of CNN (Left: accuracy, right: loss).

[...] using 19990914 build on win98 using the new account wizard if I add multiple accounts with the same server [..]

FIGURE 8
Visualization of a bug report with a short fixing time.

[...] this mean the problem was with their web page and its cool now or we still need to figure out what the actual bug was and fix that [...]

FIGURE 9
Visualization of a bug report with a long fixing time.

phenomenon such as "19990914." In contrast, bug reports with long fixing times visualized abstract terms such as "problem" and "figure" (Figure 9).

Figure 10 shows the extracted words by parts of speech, where NN, CD, JJ, IN, DT, RB, VB, VBZ, VBP, and TO indicate noun, cardinal number, adjective, preposition or subordinating conjunction, determiner, adverb, verb in the base form, verb in the third person singular present form, a verb in the non-third person singular present form, and a preposition or infinitive marker, respectively. Important words for bug reports with short bug fixing times were NN followed by CD and JJ. However, those with long bug fixing times were NN followed by IN and JJ. Additionally, CD differed significantly by fixing times.

Because CD contents were often from stack trace descriptions, whether the decision basis included a stack trace was evaluated. A stack trace gives a report of the active stack frames at a particular point during program execution. There was a stark difference by fixing time. For bug reports with short fixing times, 29.6% of all words in the CNN decision basis contained stack trace descriptions. In contrast, only 2.8% of the cases contained stack descriptions for bug reports with long fixing times. This suggests that the presence of a stack trace description may be a feature of short bug fixing times. If a reporter provides the stack trace, the developer tends to reference it. Since stack trace information in bug reports is a suitable source for bug localization (Wong et al., 2014), and this is a reasonable observation.

RQ2. What words compose the basis for CNN's decisions in bug reports? **Important words of bug reports with short bug fixing times are NN followed by CD and JJ. Those with long bug fixing times are NN followed by IN and JJ. There is a significant difference in CD by fixing time. Bug reports with short bug fixing times tend to contain stack trace descriptions, whereas those with long fixing times do not.**

4.4. RQ3: Difference of important words according to the part of speech

Important nouns (NN) and verbs (VB) were examined. The results excluding the stack trace are shown below, where the number in parentheses represents the frequency of occurrence. Figures 11, 12 show word clouds of important nouns and verbs, respectively.

Bug reports with short bug fixing times had many nouns related to the phenomenon. The most frequent word was "line" (23 times), which referred to the line number description in the stack trace. This was followed by "html" (14). This denoted the basis for CNN's decision in the text related to the bug phenomenon. For example, *"I tried opening other simple HTML files."* It also appeared as part of the URL. "Menu" (11) indicated bug phenomena such as *"after launching the app select composer under task menu."* Additionally, "window" (10), "browser" (10), and "file" (9) appeared frequently as the basis for CNNs decision.

Bug reports with long fixing times contained different nouns. The most frequent was "bug" (22), which denoted different bug

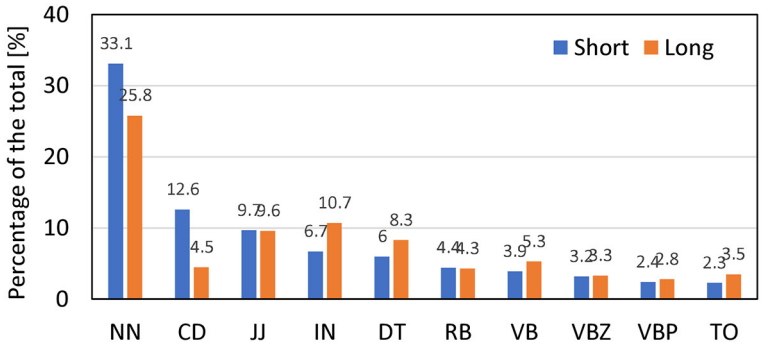

FIGURE 10
Percentage of important word by part of speech.

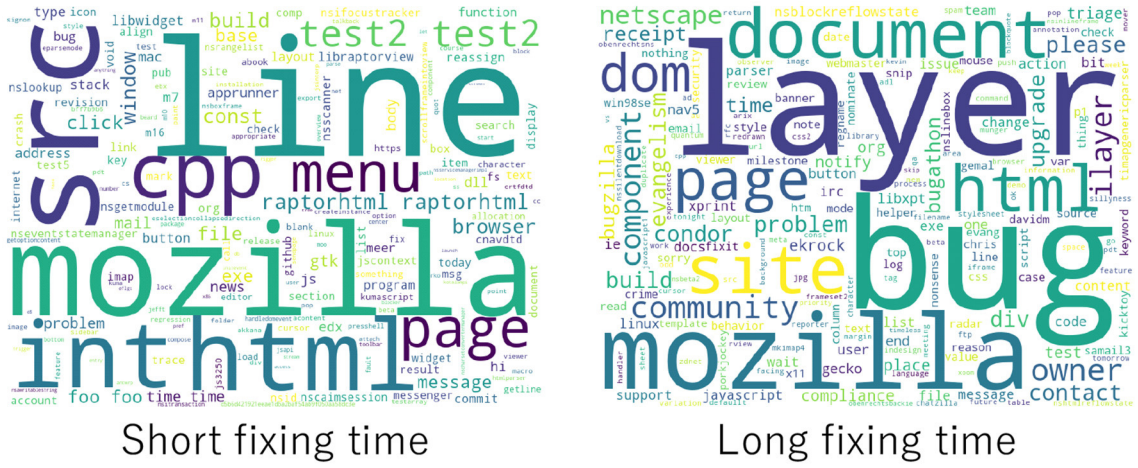

FIGURE 11
Word clouds of nouns in bug reports by fixing time.

reports, reopening, or duplicate bugs. For example, *"There is a JavaScript problem, which I saw in another bug."* The next most common noun was "document" (16), followed by "problem" (5), "place" (5), and "contact" (5). Bug denoted the phenomenon of a bug or the document layer. For example, *"This page uses all sorts of nasty document layers stuff to show hidden layers."* The word "problem" presented information. For example, "This means the problem was with their web page."

Nouns based on content were more prevalent for shorter bug fixing times. In contrast, abstract nouns were more prevalent for longer bug fixing times. The results demonstrated that abstraction level in the comments and bug fixing time was related.

Bug reports with short bug fixing times had verbs related to describing a story. The most frequent was "is" (28 times) followed by "using" (6), which appeared when providing information. For example, *"Leak found using beard's Boehm.GC."* Additionally, "need" (4), "fixed" (4), and "cause" (4) were related to short bug fixing times. For example, *"I need to implement IMAP save message to disk for us to fix this bug."*

Bug reports with long bug fixing times had verbs related to telling. Similar to bug reports with a short fixing time, the most frequent verb for bug reports with a long fixing time was "is" (33). However, the next most frequent was "moving" (13), which referred to milestones and the phenomenon. For example, *"I won't get to will refit individual milestones after moving them."* This was followed by "see" (10) when discussing phenomenon and other bugs and "works" (7), which provided information such as bug phenomena and software not working.

The results indicated that the fixing time affected the basis. For bug reports with short fixing times, more concrete words that tell a story such as identifying the cause were extracted. In contrast, more abstract words that did not advance the discussion were extracted for bug reports with long fixing times. Hence, bug reports with shorter bug fixing times discussed the bug specifically, whereas those with longer bug fixing times did not.

RQ3. Does bug fixing time influence the basis of CNN's decision? **The extracted words depend on fixing time. Concrete words tend to be extracted for bug reports with shorter fixing times, while abstract words tend to be extracted for those with long fixing times.**

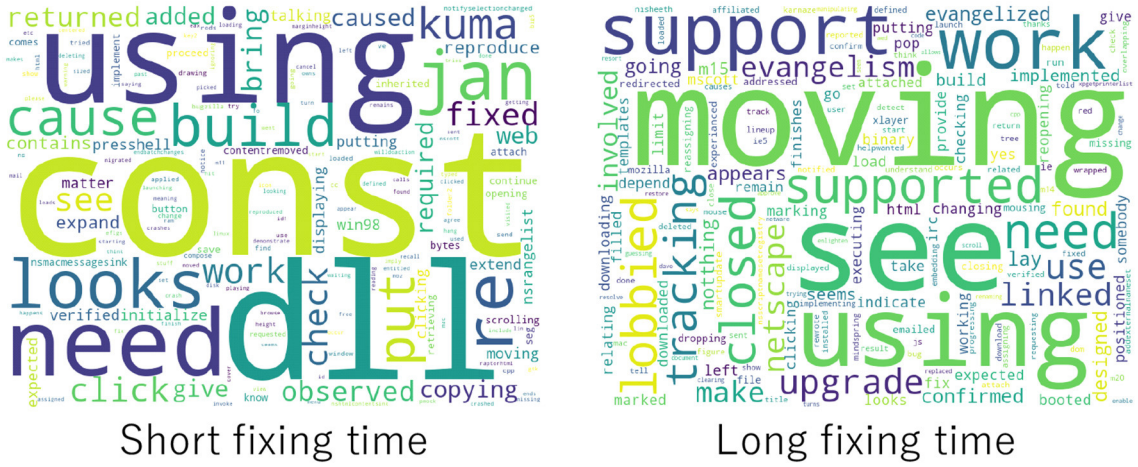

FIGURE 12
Word clouds of verbs in bug reports by fixing time.

5. Threats to validity

This study used bug report comments to classify bug reports into two groups by bug fixing time. Other factors such as bug difficulty and experience were not considered. This is a threat to internal validity.

Additionally, only closed bug reports were used. Some bug reports lacked a discussion, while others contained only one or two comments. This is another threat to internal validity because the results may change if the model learns using bug reports with a small number of comments. To address this issue, the influence of other impact factors such as maturity of the discussions and comments on feature extraction should be investigated.

The experimental results may be specific to the dataset used in our study. This is a threat to external validity. In future, other datasets should be included to clarify if similar results are generated.

6. Recommendations based on the findings

Reporters and developers should use caution when working with bug reports. To reduce the burden on developers, reporters should include details of how the bug occurred such as the stack trace. Then, developers can reference such information to more easily fix bugs.

To improve the efficiency when fixing bugs, developers should consider the level of abstraction in their comments. Abstract discussions lead to longer bug fixing times. Concrete discussions are a characteristic of bug reports with short bug fixing times. In this study, CNN employs words in concrete discussions to predict bug fixing time. Hence, word specificity should be considered when discussing a bug report.

7. Conclusion and future work

This research investigated the influence of the description and the discussion in a bug report on bug fixing time. The CNN model predicted and classified bug reports with short or long bug fixing times with a 75%–80% accuracy. The gradient-based visualization clarified the words that the model used to extract the features related to bug fixing time. Bug fixing time affected word specificity. For example, stack traces appear often in bug reports with short fixing times, whereas they rarely appear for those with long fixing times. Based on the result, reporters and developers should use caution when working with bug reports. To reduce the burden on developers, reporters should include details of how the bug occurred, such as the stack trace. Furthermore, developers should consider the level of abstraction in their comments to improve efficiency when fixing bugs. Abstract discussions lead to longer bug fixing times.

To enhance the effectiveness of our method and explore new applications to achieve effective and practical handling of bug reports, there are multiple directions for future research roughly classified into four types as follows: an examination of our method's design and settings in detail, evaluation of universality resulting in generalization of our method, comparison with other existing state-of-the-art approaches and identification of potential improvements, and incorporation of other factors.

The first is to examine the architecture of our method based on deep learning and extend it to improve the classification accuracy as well as the effectiveness of feature identification.

The second is to evaluate the robustness and universality of our method under various environments. To consider them, we plan to conduct N-fold cross-validation experiments using different N values to confirm their impact on the result. Furthermore, it is necessary to validate the universality of our method by evaluating the performance using other real-world datasets with more reports.

The third is to compare our method with approaches based on other existing state-of-the-art models and identify potential improvements and extensions of our method.

Finally, we plan to consider other factors, particularly the complexity of the bug and the maturity of the developer, for further accurate prediction and a better understanding of the important features.

Data availability statement

The original contributions presented in the study are included in the article/supplementary material, further inquiries can be directed to the corresponding author.

Author contributions

YN has handled conceptualization and methodology. All authors have contributed to the literature review and analysis, read, and agreed to the published version of the manuscript.

Acknowledgments

We thank Prof. Foutse Khomh, Prof. Yann-Gael Gueheneuc, and Mr. Qicong Liu for their assistance and useful discussions.

Conflict of interest

YN, KO, HK, and SN were employed by Hitachi, Ltd.

The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

Ahmed, H. A., Bawany, N. Z., and Shamsi, J. A. (2021). Capbug-a framework for automatic bug categorization and prioritization using NLP and machine learning algorithms. *IEEE Access* 9, 50496–50512. doi: 10.1109/ACCESS.2021.3069248

Gacitua, R., Sawyer, P., and Gervasi, V. (2010). "On the effectiveness of abstraction identification in requirements engineering," in *RE 2010, 18th IEEE International Requirements Engineering Conference*, September 27–October 1, 2010 (Sydney, NSW: IEEE Computer Society), 5–14. doi: 10.1109/RE.2010.12

Giger, E., Pinzger, M., and Gall, H. C. (2010). "Predicting the fix time of bugs," in *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering, RSSE 2010*, May 4, 2010, eds R. Holmes, M. P. Robillard, R. J. Walker, and T. Zimmermann (Cape Town, South Africa: ACM), 52–56. doi: 10.1145/1808920.1808933

Gomes, L., Cortes, M., and Torres, R. (2022). Bert-based feature extraction for long-lived bug prediction in floss: a comparative study. *SSRN* 1–31. doi: 10.2139/ssrn.4166555

Gomes, L., Cortes, M., and Torres, R. (2023). Bert- and tf-idf-based feature extraction for long-lived bug prediction in floss: a comparative study. *Inf. Softw. Technol.* 160, 1–12. doi: 10.1016/j.infsof.2023.107217

Guo, J., Cheng, J., and Cleland-Huang, J. (2017). "Semantically enhanced software traceability using deep learning techniques," in *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017*, May 20–28, 2017, eds S. Uchitel, A. Orso, and M. P. Robillard (Buenos Aires, Argentina: IEEE/ACM), 3–14. doi: 10.1109/ICSE.2017.9

Habayeb, M., Murtaza, S. S., Miranskyy, A. V., and Bener, A. B. (2018). On the use of hidden markov model to predict the time to fix bugs. *IEEE Trans. Software Eng.* 44, 1224–1244. doi: 10.1109/TSE.2017.2757480

Han, Z., Li, X., Xing, Z., Liu, H., and Feng, Z. (2017). "Learning to predict severity of software vulnerability using only vulnerability description," in *2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017*, September 17–22, 2017 (Shanghai, China: IEEE Computer Society), 125–136. doi: 10.1109/ICSME.2017.52

He, J., Xu, L., Yan, M., Xia, X., and Lei, Y. (2020). "Duplicate bug report detection using dual-channel convolutional neural networks," in *ICPC '20: 28th International Conference on Program Comprehension*, July 13–15, 2020 (Seoul, Republic of Korea: ACM), 117–127. doi: 10.1145/3387904.3389263

Hey, T., Keim, J., Koziolek, A., and Tichy, W. F. (2020). "Norbert: Transfer learning for requirements classification," in *28th IEEE International Requirements Engineering Conference, RE 2020*, T. D. Breaux, A. Zisman, S. Fricker, and M. Glinz, August 31–September 4, 2020 (Zurich, Switzerland: IEEE), 169–179. doi: 10.1109/RE48521.2020.00028

Hirakawa, R., Tominaga, K., and Nakatoh, Y. (2020). "Study on automatic defect report classification system with self attention visualization," in *2020 IEEE International Conference on Consumer Electronics (ICCE)*, January 4–6, 2020 (Las Vegas, NV: IEEE), 1–2. doi: 10.1109/ICCE46568.2020.9043062

Isotani, H., Washizaki, H., Fukazawa, Y., Nomoto, T., Ouji, S., Saito, S., et al. (2021). "Duplicate bug report detection by using sentence embedding and fine-tuning," in *IEEE International Conference on Software Maintenance and Evolution, ICSME 2021*, September 27–October 1, 2021 (Luxembourg: IEEE), 535–544. doi: 10.1109/ICSME52107.2021.00054

Kalchbrenner, N., Grefenstette, E., and Blunsom, P. (2014). "A convolutional neural network for modelling sentences," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (Baltimore, MD: Association for Computational Linguistics),655–665. doi: 10.3115/v1/P14-1062

Kowsari, K., Jafari Meimandi, K., Heidarysafa, M., Mendu, S., Barnes, L., Brown, D., et al. (2019). Text classification algorithms: a survey. *Information* 10, 150. doi: 10.3390/info10040150

Lawrence, S., Giles, C. L., Tsoi, A. C., and Back, A. D. (1997). Face recognition: a convolutional neural-network approach. *IEEE Trans. Neural Netw.* 8, 98–113. doi: 10.1109/72.554195

Lee, Y., Lee, S., Lee, C., Yeom, I., and Woo, H. (2020). Continual prediction of bug-fix time using deep learning-based activity stream embedding. *IEEE Access* 8, 10503–10515. doi: 10.1109/ACCESS.2020.2965627

Li, J., He, P., Zhu, J., and Lyu, M. R. (2017). "Software defect prediction via convolutional neural network," in *2017 IEEE International Conference on Software Quality, Reliability and Security, QRS 2017*, July 25–29, 2017 (Prague, Czech Republic: IEEE), 318–328. doi: 10.1109/QRS.2017.42

Marks, L., Zou, Y., and Hassan, A. E. (2011). "Studying the fix-time for bugs in large open source projects," in *Proceedings of the 7th International Conference on Predictive Models in Software Engineering, PROMISE 2011*, eds T. Menzies, September 20–21, 2011 (Banff, AB: ACM), 11. doi: 10.1145/2020390.2020401

Minaee, S., Kalchbrenner, N., Cambria, E., Nikzad, N., Chenaghlu, M., Gao, J., et al. (2022). Deep learning-based text classification: a comprehensive review. *ACM Comput. Surv.* 54, 62:1–62:40. doi: 10.1145/3439726

Neysiani, B. and Morteza, B. (2020). "Automatic duplicate bug report detection using information retrieval-based versus machine learning-based approaches," in *2020 6th International Conference on Web Research, ICWR* 2020 (Tehran: IEEE), 288–293. doi: 10.1109/ICWR49608.2020.9122288

Noyori, Y., Washizaki, H., Fukazawa, Y., Kanuka, H., Ooshima, K., Nojiri, S., et al. (2021a). What are the features of good discussions for shortening bug fixing time? *IEICE Trans. Inf. Syst.* 104-D, 106–116. doi: 10.1587/transinf.2020MPP0007

Noyori, Y., Washizaki, H., Fukazawa, Y., Kanuka, H., Oshima, K., Tsuchiya, R., et al. (2018). "Improved searchability of bug reports using content-based labeling with machine learning of sentences," in *Knowledge-Based Software Engineering: 2018, Proceedings of the 12th Joint Conference on Knowledge-Based Software Engineering (JCKBSE 2018)*, eds M. Virvou, F. Kumeno, and K. Oikonomou (Corfu, Greece: Springer), 75–85. doi: 10.1007/978-3-319-97679-2_8

Noyori, Y., Washizaki, H., Fukazawa, Y., Ooshima, K., Kanuka, H., Nojiri, S., et al. (2021b). "Extracting features related to bug fixing time of bug reports by deep learning and gradient-based visualization," in *Proceedings of the IEEE International Conference on Artificial Intelligence and Computer Applications, ICAICA, Online*, June 28–30, 2021 (Dalian: IEEE Computer Society), 402–407. doi: 10.1109/ICAICA52286.2021.9498236

Palacio, D. N., McCrystal, D., and Moran, K. Bernal-Cárdenas, C., Poshyvanyk, D., Shenefiel, C. (2019). "Learning to identify security-related issues using convolutional neural networks," in *2019 IEEE International Conference on Software Maintenance and Evolution, ICSME 2019*, September 29–October 4, 2019 (Cleveland, OH: IEEE), 140–144. doi: 10.1109/ICSME.2019.00024

Panjer, L. D. (2007). "Predicting eclipse bug lifetimes," in *Fourth International Workshop on Mining Software Repositories, MSR 2007 (ICSE Workshop)*, May 19–20, 2007 (Minneapolis, MN: IEEE Computer Society), 29. doi: 10.1109/MSR.2007.25

Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., Batra, D., et al. (2017). "Grad-cam: visual explanations from deep networks via gradient-based localization," in *IEEE International Conference on Computer Vision, ICCV 2017*, October 22–29, 2017 (Venice, Italy: IEEE Computer Society), 618–626. doi: 10.1109/ICCV.2017.74

Shokripour, R., Kasirun, Z. M., Zamani, S., and Anvik, J. (2012). "Automatic bug assignment using information extraction methods," in *Proceedings - 2012 International Conference on Advanced Computer Science Applications and Technologies, ACSAT* 2012 (Kuala Lumpur), 144–149. doi: 10.1109/ACSAT.2012.56

Wong, C., Xiong, Y., Zhang, H., Hao, D., Zhang, L., Mei, H., et al. (2014). "Boosting bug-report-oriented fault localization with segmentation and stack-trace analysis," in *30th IEEE International Conference on Software Maintenance and Evolution*, September 29–October 3, 2014 (Victoria, BC: IEEE Computer Society), 181–190. doi: 10.1109/ICSME.2014.40

Youm, K. C., Ahn, J., Kim, J., and Lee, E. (2015). "Bug localization based on code change histories and bug reports," in *2015 Asia-Pacific Software Engineering Conference, APSEC 2015*, eds J. Sun, Y. R. Reddy, A. Bahulkar, and A. Pasala, December 1–4, 2015 (New Delhi, India: IEEE Computer Society), 190–197. doi: 10.1109/APSEC.2015.23

Yusop, N. S. M., Grundy, J. C., and Vasa, R. (2016). "Reporting usability defects: do reporters report what software developers need?," in *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering, EASE 2016*, eds S. Beecham, B. A. Kitchenham, and S. G. MacDonell, June 01–03, 2016 (Limerick, Ireland: ACM), 38:1–38:10. doi: 10.1145/2915970.2915995

Zhang, F., Khomh, F., Zou, Y., and Hassan, A. E. (2012). "An empirical study on factors impacting bug fixing time," in *19th Working Conference on Reverse Engineering, WCRE 2012*, October 15–18, 2012 (Kingston, ON: IEEE Computer Society), 225–234. doi: 10.1109/WCRE.2012.32

Zhang, H., Gong, L., and Versteeg, S. (2013). "Predicting bug-fixing time: an empirical study of commercial software projects," in *35th International Conference on Software Engineering, ICSE '13*, eds D. Notkin, B. H. C. Cheng, and K. Pohl, May 18–26, 2013 (San Francisco, CA: IEEE Computer Society), 1042–1051. doi: 10.1109/ICSE.2013.6606654

Zhang, T., Chen, J., Jiang, H., Luo, X., and Xia, X. (2017). "Bug report enrichment with application of automated fixer recommendation," in *Proceedings of the 25th International Conference on Program Comprehension, ICPC 2017*, eds G. Scanniello, D. Lo, and A. Serebrenik, May 22–23, 2017 (Buenos Aires, Argentina: IEEE Computer Society), 230–240. doi: 10.1109/ICPC.2017.28

Zhang, Y. and Wallace, B. (2017). "A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification," in *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)* (Taipei, Taiwan: Asian Federation of Natural Language Processing), 253–263.

Zimmermann, T., Premraj, R., Bettenburg, N., Just, S., Schröter, A., and Weiss, C. (2010). What makes a good bug report? *IEEE Trans. Softw. Eng.* 36, 618–643. doi: 10.1109/TSE.2010.63