Mining the Software Engineering Forums: What's New and What's Left

Wei Yuan[1], Peng Wang[2], Yue Guo[1], Linyang He[1], and Tieke He[1](✉)

[1] State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China
hetieke@gmail.com

[2] Jiangsu Tongxingbao Intelligent Transportation Technology Co., Ltd., Nanjing, China

Abstract. Software maintenance is an important part of the software life cycle. People use bug tracking systems to collect bugs in the system. There are a lot of bug reports in open source software. Researchers conducted a series of studies on these reports, such as automatically determining whether two bug reports were duplicates. This article provides a detailed survey of the researchers' analysis of bug reports. In this paper, we conduct a comprehensive survey of the works concerning the mining of the software engineering forums. Specifically, we formulate these works in a three-dimensional style, i.e., we classify these studies according to the data formats they used, the methodology they are adopting, and most importantly, the questions they are dealing with. With this three dimensional partition, it can be clearly known what has been done, and what is left, along with the question, say, why left? To go further, beyond this three-dimensional partition, we are seeking to add research space through new data, novel techniques, and upcoming research questions.

Keywords: Software maintenance · Data mining · Bug report

1 Introduction

It is difficult for humans to develop software without any problems. So, finding and fixing bugs is an important process in the software life cycle. People use bug tracking systems to collect software system errors discovered by developers, testers, and end-users. The most commonly used bug tracking system is Bugzilla[1]. Many open-source projects use Bugzilla to help them manage their projects. For example, eclipse receives a lot of bug reports every day[2]. But everything has two sides. On the one hand, a lot of reports can help us improve the quality of software, however, on the other hand, handling these reports manually is a very time consuming task. If we can't extract useful value information from these bug reports, then more data doesn't make any sense.

[1] https://www.bugzilla.org/.
[2] https://bugs.eclipse.org/bugs/.

G. Wang et al. (Eds.): WISA 2020, LNCS 12432, pp. 513–524, 2020.
https://doi.org/10.1007/978-3-030-60029-7_46

Due to a large number of bug reports, it is unrealistic to rely solely on people to deal with them. People are gradually proposing more and more analytical methods to deal with bug reports. In this article, we will investigate the different problems that researchers have studied in existing erroneous data sets. We want to express what technologies people have used so far to solve the various problems in the bug report.

In order to better review the existing work and look forward to the future work, we sort out the previous work of the researchers from the three dimensions: problem, data and technology. From a problem perspective, some people focus on the detection of bugs, they want to analyze the existing bug reports and extract the characteristics of the bugs. When they receive a new bug report, let the machine automatically determine if this is a real error, whether it is a duplicate of the bug in the existing bug library, and find a bug similar to this bug to solve this problem faster. Some researchers want to automatically identify the severity and priority of new bugs through existing bug reports so that developers can prioritize the most problem-solving issues and do more meaningful things in a limited amount of time. There is a bug in the system, usually because some files are written. Some researchers want to find the relationship between bugs and source files through the existing repair experience, let people locate bugs and fix bugs faster. Some researchers believe that different people have different ability to solve bugs in different fields. They want to learn the distribution of existing bugs and automatically recommend the most suitable developers to solve problems when new bugs occur.

From a data perspective, most of the work already done is based on Bugzilla collecting bug reports. However, the fields that different researchers pay attention to are not the same. Some people only pay attention to the text information in the bug report, such as description, summary field. Some people think that structured information also contains important information. They not only consider text information but also consider structured information such as priorities and components. Still others believe that just analyzing bug reports is not enough. They introduce external data combined with bug reports to solve problems. How to reasonably combine different types of information has always been a difficult point. From a technical point of view, most of the work is based on traditional information retrieval techniques, machine learning and deep learning techniques. In recent years, with the development of big data and the significant improvement of computing power, people think that "big data + complex model" is a better choice. Therefore, neural networks and deep learning have become more and more popular, and they have demonstrated their capabilities in various fields. More and more people are trying to use neural networks instead of traditional information retrieval and basic machine learning algorithms to solve different problems.

2 Bug Report Problem Classification

In this survey, we mainly summarize the problems that others have studied on the bug report from the perspective of the problem and analyze the data and

technology they use. We divide the existing research related to the bug report into five categories: bug detection, bug level determination, bug location, bug developer recommendation, and some other issues.

In terms of bug detection, we mainly consider three sub-problems: bug classification, to determine whether a report is a bug; Duplicate bug detection to determine if the two bug reports describe the same problem; Similar bug detection to determine if two bugs are of the same type. The bug level determination is mainly divided into two sub-problems: bug severity prediction and bug priority prediction. The bug location and bug developer recommendations are two separate and widely studied issues. By dividing the bug report problem into different categories, it is easy to know the hot point research area in recent years and conclude future tendencies.

3 Bug Detection

There are three main problems with bug detection. The first question is whether the problem described in the bug report is about the bug. Users not only submit a bug to the tracking system, they also mention other requirements, such as how to make the system more convenient. Thus, we need to identify the real bug and then solve it. The second question is about duplication. Many reports submitted by users refer to the same bug but with different descriptions, we need to be able to determine whether the two bug reports are about the same problem. The third problem is to identify similar bugs. We know that if the two errors are very similar, then the reasons for them may be very similar. By recommending similar bug reports to developers, it may be faster to locate errors and fix bugs.

3.1 Bug Reports Classification

There are a large number of reports that are actually misclassified. Manually classifying bug reports is a very time consuming task. [1] used 90 days to manually sort over 7,000 error reports. Therefore, it is necessary to classify reports automatically.

Researchers firstly used the text field in the bug report to extract features and use this to determine if the new report was a bug. [2] applied topic modeling to the corpus of pre-processed bug reports, and then classified bug reports using decision trees, naive Bayes classifiers, and logistic regression. Experiments implicate that the topic-based model outperforms than the word-based model, and the naive Bayesian model is better than the other two in classification.

Some researchers believe that structured information in the bug report can help judge whether a report is a bug. [3] used a hierarchical Dirichlet process (HDP) and clustering to classify bug reports. The bug report is projected into the topic vector space, then clustering method is utilized to aggregate the bug reports and tag the categories. How to better combine structured and unstructured data has always been an important issue. Some people [4] proposed a hybrid approach to classify error reports. They used the text mining method to extract features

from the report summary, and then used the data mining method to combine the structured information features in the error report with the previous stage text features, finally used the Bayesian classifier to predict.

3.2 Duplicate Bug Report

In 2018, [5] found that the proportion of duplicate bugs reached 20% on the bug repositories of Mozilla Core, Firefox and so on. These duplicated reports may be solved by developers for many times, resulting in waste of human resources. So far, the bug tracking system can not detect duplicate bugs automatically when collecting bugs [6]. The work of detecting duplicate bugs can be divided into two categories: One is to apply natural language processing (NLP) techniques to unstructured textual information, such as bug title and bug summary; Another approach focuses on execution information in bug reports.

Duplicate Bug Detection Based on NLP. [7] used BM25 for term weighting to transform bug reports into vector space. They found that the right term weighting is critical for detecting duplicate bug reports.

There are many challenges in using text information. As each person has different speaking habits, different words may be used to express the same concept, so only use text matching is not enough. Hence, it is necessary to analyze the semantic of bug report. [8] treated each bug report as a text document and used it to train word embedding models [9]. Using the trained word embedding model, They converted the error report into a vector and further trained the deep neural network above these bug report vectors to understand the distribution of duplicate bug reports and non-repeated bug reports. In addition to the above method, there are some other attempts. [10] proposed a combination of Latent Dirichlet Allocation (LDA) and word embedding method to determine whether it is a duplicate bug report. The idea of this approach is to use LDA's higher recall rate to first exclude the most dissimilar bug reports, and then use the word embedding model in the remaining reports to calculate the similarity among reports.

Duplicate Bug Detection Based on Execution Information. The bug execution information describes the context in which the bug occurred. The context of repeated bugs is the same. [11] proposed a repetitive error detection involving both execution information and natural language information. [12] considered to utilize domain knowledge and context of software. In the Android bug tracking system experiment, they found that the detection of bug reports can be improved by considering keywords in the Android domain. Some people think that the more features of a bug report you have, the more detailed you can portray a bug. [13] defined 25 features in the bug report as the basis for the classification, most of which were generated by the TakeLab system, and then used the SVM training model to classify the bugs. The topic model enables efficient semantic analysis and text mining. [14] proposed a novel duplication

detection method based on the topic model. They combined the similarity of each report in the topic space with the similarity of the classified information of each report to predict duplication.

Deep learning methods also be involved in this area. [15] proposed a search and classification model combining CNN and LSTM [16]to solve the problem of repeated bugs. They used the vanilla single layer neural network to handle structured information in bug reports, used LSTM to handle short descriptions, used CNN to process long descriptions, and finally combined them to learn bug reporting features. Some people [5] proposed to use word embedding and Convolution Neural Networks to calculate the similarity between bug reports, because this not only pays attention to textual similarities but also achieves semantic similarity. This method not only considers the textual information in the report, but also combines domain-specific features (i.e., Component, Create time and Priority, etc.) to better detect duplicate bug reports. Over time, more and more new models have been proposed for specific problems. [17] used stack traces and hidden Markov models to automatically detect duplicate bug reports. Based on their research, they recognized the obvious benefits of using stack trace information. They believe that this information can improve the accuracy of the detection of repeated bug reports. They used recall rate and Mean Average Precision (MAP) to evaluate their models on Firefox and GNOME datasets and found better results than baseline models.

3.3 Similar Bug Report

Similar errors mean that bugs in several bug reports are related to common code files. Unlike duplicate bug reports, we generally think that two reports are similar reports when they have more than 50% modified common files. By recommending similar bug reports to developers, we can help them locate the cause of the error faster and solve new bugs efficiently.

[18] combined traditional information retrieval technology and word embedding technology, and considered the title, description, and other component information in the bug report to recommend similar reports to developers. They experimented with similar bug recommendations, and their approach has better performance than NextBug [19]. Inspired by this, [20] proposed a new method for using document embedding models in order to further improve the performance of the method. They added a new document embedding vector component to the existing three components. This component focuses on mining the potential relationships between the two bug reports at the document level for better results.

4 Bug Level Determination

4.1 Bug Reports Severity

The bug report generally includes a severity, which helps the developer to resolve the serious error first. The severity is divided into crashes, errors, low efficiency

and minors. Automatically detecting the degree of severity can benefit bug report processing, letting high severity bug reports be processed preferentially. [21] extracted the concept word in the bug report to construct the concept profile (CP). When a new bug report is encountered, they calculate the degree of similarity between the report and the CP severity concept to determine the severity of the bug. How to determine the severity of the profile requires people to do it manually. [22] use unsupervised methods to determine whether the severity of the bug report is correctly assigned. They used a Gaussian mixture model to group similar bug reports, then assigned severity labels to grouped bug reports, and finally used supervised algorithms to predict the severity of unmarked bug reports.

Just thinking about textual information is not enough. [23] proposed a nearest neighbor method based on information retrieval to predict the severity of bugs. They used the extended BM25 document similarity function to select the k reports that are most similar to the new bug report, and then predicted the severity of the new bug based on the severity of the k reports. In addition to considering the text information in the bug report, they also considered structured information like product and component.

4.2 Bug Reports Prioritization

[24] proposed to use different machine learning algorithms such as Naïve Bayes, decision trees, and random forests to predict the priority of reported bugs. They experimented on two feature sets. The first feature set is based on the textual description of the error report. The second feature set is based on the predefined metadata of the bug report. Experiments showed that the classification results of random forests and decision trees are better than Naïve Bayes, and the results of the second feature set are better than the first feature set. [25] thought that previous researchers did not take into account the reporter's sentiment when predicting the priority of bug reports. In the bug report, if the submitter's description is very anxious, the severity and priority of the bug may be high. Therefore, they extracted features from the bug report, then used the sentiment words involved in the bug report summary to calculate the sentiment value of each bug report, and then combined the two to train and predict the priority of the bug report. Rich, unstructured information in bug reports was also involved. [26] extracted the temporal, textual, author, related-report, severity, and product in the bug report as features, and then used the linear regression model to determine the priority of the report. They defined the priority of five bug reports and then used the thresholding approach to solve the problem of data imbalance.

5 Bug Localization

To solve the bug, the system developer needs to locate the source file that caused the bug which would be difficult especially in a huge system. The biggest challenge in auto-locating bugs is the mismatch between the terms used in the bug

report and the terms used in the source file. There are lots of methods for solving this issue. One of the directions is analyzing the source code file. [27] believed that structured information based on code structure such as class name and a method name can help us locate bugs better. Their method only required source code and error reporting. However, the terms used in the error report used to describe the error may not be the same as the terms used in the source file. [28] combined DNN and information retrieval (IR) techniques to locate error files associated with bugs. They used information retrieval techniques to calculate textual similarities between error reports and source files. DNN is used to link the specific terms in the bug report to the terms in the source file. [29] found that if the error report lacks rich and structured information, the information retrieval technology often does not work well, and too much stack tracking information does not help the positioning.

Some researchers believed that considering the version history can better locate potential error locations. [30] proposed AmaLgam, a model that combines historical data, similar reports, and structural information to locate files related to bugs and achieved good performance. To help better understand existing code, researchers use information retrieval techniques to map bug reports to associated code units. [31] proposed a variant of 15 vector space models based on tf-idf to form a new composite model. They used the VSM model and AmaLgam [30] to calculate the weighted sum of suspicious files to locate bug files.

6 Bug Developer Recommendation

When we encounter a new bug, which developer should I assign to fix it? In general, we can randomly assign bugs to the developer, but it is deficient. Generally, developers have their own expertise area, so it is better to recommend bug reports that belongs to this area for them to fix up. Researchers attempt to address this issue by analyzing different kinds of data, such as text information, structured information and developer profile, and so on.

[32] believed that most of the previous work focused only on open source projects. They used convolutional neural networks and word embedding to build auto-recommended developers to fix bugs and apply the technology to industrial projects and open source projects. They believed that there are two main challenges. The first challenge is that in multinational companies whose native language is not English, bug reports often appear in their native language and English. The second challenge is that industrial projects are different from open source projects, and there may be many specific terms in specific fields. They mainly extracted the two text fields of description and summary in the bug report as features. They also proposed the idea of manual and automated classification cooperation and introduced the experience used in the industrial development environment.

Textual information in bug reports alone often does not yield satisfactory results. [33] introduced a highly scalable recommendation system for bug reporting assignments. In addition to considering the textual information of the report,

they also used structured information such as component id, product id, and bug severity as feature data. They used convolutional neural networks and recurrent neural networks as deep learning classifiers. At the same time, they also made certain restrictions on developers. They believe that only developers who are still active in the project should be assigned bug fixes. They believe that only developers who have been fixing bugs for a while can be considered an active developer. They opened up further research directions in optimizing training speed and predictive performance.

Summarizing the bugs that each developer has fixed in the past is a good way to portray developers. [34] proposed to create an activity profile for the history of all activities of all users in the bug tracking system, then model it according to this file, and then recommend the appropriate developer to solve the bug through this model. Through this file, we can probably know the role of the developer in this system and the areas of expertise. Although we can better distribute bugs to developers through configuration files, it takes time to mine the configuration files for each developer's historical data. [35] proposed a new method called DevRec, which consists of two types of analysis, bug reports based analysis and developer based analysis. The bug-based analysis is mainly to find bugs similar to the newly collected bugs from the bug repository. By analyzing the bug fixers, he can found potential fix developers for new bugs. They converted the features in the bug report into vectors to calculate the similarity between the two reports. The developer-based analysis measures the distance between the bug report and the developer, correlating the developer with the characteristics of the bug report. They combined these two analyses for optimal performance. [36] proposed a unified model based on learning ranking technology, which combines activity-based technology to find out which developers have solved similar bugs and location-based techniques to find the right developer for the bug location.

7 Other Problems

7.1 Generating Bug Fixes

Although the bug report tells the developer that there is a defect in the system, it may not be able to fix the defect due to the lack of a development environment, and the defect remains in the system. [37] proposed a method called R2Fix, which automatically generates bug fixes from bug reports. They chose buffer overflow, null pointer error and memory leak to evaluate the proposed method, because the repair methods of these three types of errors are relatively simple, and the repair mode can be found. When R2Fix received a new bug report, it analyzed the bug report, determined which error belongs to the above three types of errors, and finally generates a possible patch to fix the bug. In the verification experiment, R2Fix automatically generated the correct patch for 57 errors with an accuracy of 71.3%, and it also found potential errors that the tester did not find. Due to the difficulty of automatically generating bug fix, there is still a long way to go to generalize automatic patch generation.

7.2 Automatic Vulnerability Recognition

Automatically identify potential bugs will greatly improve the efficiency of software maintenance. In order to find unrecognized vulnerabilities in the open-source library and fix it, Some people [38] proposed an automatic vulnerability identification system. They used a variety of machine learning classifiers as basic classifiers to extract features from bug report submissions and reports themselves and automatically discovered unrecognized errors in submitted reports through natural language processing and machine learning techniques. However, because of the complexity of this issue, existing methods perform not well. Future work will continue exploring new methods to solve this problem.

7.3 Bug Report Summarization

In order to help developers quickly understand the information in the bug report, [39] proposed a two-layer semantic model (TSM) to extract important information from the report. They first used the extended NR (ENR) model to preserve the sentences with important semantics in the report, then used BRC (Bug Report Classifier) to extract the text features from these sentences, and finally used the logistic regression training model to select the sentences with high scores to generate the abstract of the article. [40] explored the use of deep neural networks to generate a summary of bug reports. They used bug report preprocessing, unsupervised network training and summary generation to assign scores to sentences in bug reports, and then dynamically selected sentences with high scores to generate summaries.

8 Conclusion - What's the Outlook?

We have presented a comprehensive survey of bug report, categorizing current bug report tasks based on problem, data and technology and summarizing the current situation for each tasks. What's the next for bug report? We end with future potential directions by applying past insights to the current situation. Firstly, different types of data will be used to better analyze bug reports; Then, advanced models will be invented to better address specific problems; Last but not least, the efficiency and practicability of methods will be considered.

Acknowledgment. This work was partially supported by the Fundamental Research Funds for the Central Universities (14380023), and the Equipment Development Department Pre-Research Foundation of China (31511110310).

References

1. Herzig, K., Just, S., Zeller, A.: It's not a bug, it's a feature: how misclassification impacts bug prediction. In: Proceedings of the 2013 International Conference on Software Engineering, pp. 392–401. IEEE Press (2013)

2. Pingclasai, N., Hata, H., Matsumoto, K.I.: Classifying bug reports to bugs and other requests using topic modeling. In: 2013 20th Asia-Pacific Software Engineering Conference (APSEC), vol. 2, pp. 13–18. IEEE (2013). https://doi.org/10.1109/APSEC.2013.105
3. Limsettho, N., Hata, H., Monden, A., Matsumoto, K.: Automatic unsupervised bug report categorization. In: 2014 6th International Workshop on Empirical Software Engineering in Practice, pp. 7–12. IEEE (2014). https://doi.org/10.1109/IWESEP.2014.8
4. Zhou, Y., Tong, Y., Gu, R., Gall, H.: Combining text mining and data mining for bug report classification. J. Softw.: Evol. Process **28**(3), 150–176 (2016). https://doi.org/10.1109/ICSME.2014.53
5. Xie, Q., Wen, Z., Zhu, J., Gao, C., Zheng, Z.: Detecting duplicate bug reports with convolutional neural networks. In: 2018 25th Asia-Pacific Software Engineering Conference (APSEC), pp. 416–425. IEEE (2018). https://doi.org/10.1109/APSEC.2018.00056
6. Sun, C., Lo, D., Wang, X., Jiang, J., Khoo, S.C.: A discriminative model approach for accurate duplicate bug report retrieval. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1, pp. 45–54. ACM (2010). https://doi.org/10.1145/1806799.1806811
7. Yang, C.Z., Du, H.H., Wu, S.S., Chen, X.: Duplication detection for software bug reports based on BM25 term weighting. In: 2012 Conference on Technologies and Applications of Artificial Intelligence, pp. 33–38. IEEE (2012). https://doi.org/10.1109/TAAI.2012.20
8. Budhiraja, A., Dutta, K., Reddy, R., Shrivastava, M.: DWEN: deep word embedding network for duplicate bug report detection in software repositories. In: Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings, pp. 193–194. ACM (2018). https://doi.org/10.1145/3183440.3195092
9. Xia, C., He, T., Wan, J., Wang, H.: Ensemble methods for word embedding model based on judicial text. In: Ni, W., Wang, X., Song, W., Li, Y. (eds.) WISA 2019. LNCS, vol. 11817, pp. 309–318. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30952-7_31
10. Budhiraja, A., Reddy, R., Shrivastava, M.: Poster: LWE: LDA refined word embeddings for duplicate bug report detection. In: 2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion), pp. 165–166. IEEE (2018)
11. Wang, X., Zhang, L., Xie, T., Anvik, J., Sun, J.: An approach to detecting duplicate bug reports using natural language and execution information. In: Proceedings of the 30th International Conference on Software Engineering, pp. 461–470. ACM (2008). https://doi.org/10.1145/1368088.1368151
12. Alipour, A., Hindle, A., Stroulia, E.: A contextual approach towards more accurate duplicate bug report detection. In: 2013 10th Working Conference on Mining Software Repositories (MSR), pp. 183–192. IEEE (2013). https://doi.org/10.1109/MSR.2013.6624026
13. Lazar, A., Ritchey, S., Sharif, B.: Improving the accuracy of duplicate bug report detection using textual similarity measures. In: Proceedings of the 11th Working Conference on Mining Software Repositories, pp. 308–311. ACM (2014). https://doi.org/10.1145/2597073.2597088
14. Zou, J., Xu, L., Yang, M., Yan, M., Yang, D., Zhang, X.: Duplication detection for software bug reports based on topic model. In: 2016 9th International Conference on Service Science (ICSS), pp. 60–65. IEEE (2016)

15. Deshmukh, J., Podder, S., Sengupta, S., Dubash, N., et al.: Towards accurate duplicate bug retrieval using deep learning techniques. In: 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 115–124. IEEE (2017). https://doi.org/10.1109/ICSS.2016.16
16. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997). https://doi.org/10.1162/neco.1997.9.8.1735
17. Ebrahimi, N., Trabelsi, A., Islam, M.S., Hamou-Lhadj, A., Khanmohammadi, K.: An HMM-based approach for automatic detection and classification of duplicate bug reports. Inf. Softw. Technol. **113**, 98–109 (2019). https://doi.org/10.1016/j.infsof.2019.05.007
18. Yang, X., Lo, D., Xia, X., Bao, L., Sun, J.: Combining word embedding with information retrieval to recommend similar bug reports. In: 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), pp. 127–137. IEEE (2016). https://doi.org/10.1109/ISSRE.2016.33
19. Rocha, H., Valente, M.T., Marques-Neto, H., Murphy, G.C.: An empirical study on recommendations of similar bugs. In: 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), vol. 1, pp. 46–56. IEEE (2016). https://doi.org/10.1109/SANER.2016.87
20. Hu, D., et al.: Recommending similar bug reports: a novel approach using document embedding model. In: 2018 25th Asia-Pacific Software Engineering Conference (APSEC), pp. 725–726. IEEE (2018). https://doi.org/10.1109/APSEC.2018.00108
21. Zhang, T., Yang, G., Lee, B., Chan, A.T.: Predicting severity of bug report by mining bug repository with concept profile. In: Proceedings of the 30th Annual ACM Symposium on Applied Computing, pp. 1553–1558. ACM (2015). https://doi.org/10.1145/2695664.2695872
22. Pushpalatha, M., Mrunalini, M.: Predicting the severity of open source bug reports using unsupervised and supervised techniques. Int. J. Open Source Softw. Process. (IJOSSP) **10**(1), 1–15 (2019). https://doi.org/10.4018/IJOSSP.2019010101
23. Tian, Y., Lo, D., Sun, C.: Information retrieval based nearest neighbor classification for fine-grained bug severity prediction. In: 2012 19th Working Conference on Reverse Engineering, pp. 215–224. IEEE (2012). https://doi.org/10.1109/WCRE.2012.31
24. Alenezi, M., Banitaan, S.: Bug reports prioritization: which features and classifier to use? In: 2013 12th International Conference on Machine Learning and Applications, vol. 2, pp. 112–116. IEEE (2013). https://doi.org/10.1109/ICMLA.2013.114
25. Umer, Q., Liu, H., Sultan, Y.: Emotion based automated priority prediction for bug reports. IEEE Access **6**, 35743–35752 (2018). https://doi.org/10.1109/ACCESS.2018.2850910
26. Tian, Y., Lo, D., Sun, C.: Drone: predicting priority of reported bugs by multi-factor analysis. In: 2013 IEEE International Conference on Software Maintenance, pp. 200–209. IEEE (2013). https://doi.org/10.1109/ICSM.2013.31
27. Saha, R.K., Lease, M., Khurshid, S., Perry, D.E.: Improving bug localization using structured information retrieval. In: 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 345–355. IEEE (2013). https://doi.org/10.1109/ASE.2013.6693093
28. Lam, A.N., Nguyen, A.T., Nguyen, H.A., Nguyen, T.N.: Combining deep learning with information retrieval to localize buggy files for bug reports (n). In: 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 476–481. IEEE (2015). https://doi.org/10.1109/ASE.2015.73

29. Rahman, M.M., Roy, C.: Poster: improving bug localization with report quality dynamics and query reformulation. In: 2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion), pp. 348–349. IEEE (2018)
30. Wang, S., Lo, D.: Version history, similar report, and structure: putting them together for improved bug localization. In: Proceedings of the 22nd International Conference on Program Comprehension, pp. 53–63. ACM (2014). https://doi.org/10.1145/2597008.2597148
31. Wang, S., Lo, D., Lawall, J.: Compositional vector space models for improved bug localization. In: 2014 IEEE International Conference on Software Maintenance and Evolution, pp. 171–180. IEEE (2014). https://doi.org/10.1109/ICSME.2014.39
32. Lee, S.R., Heo, M.J., Lee, C.G., Kim, M., Jeong, G.: Applying deep learning based automatic bug triager to industrial projects. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, pp. 926–931. ACM (2017). https://doi.org/10.1145/3106237.3117776
33. Florea, A.-C., Anvik, J., Andonie, R.: Parallel implementation of a bug report assignment recommender using deep learning. In: Lintas, A., Rovetta, S., Verschure, P.F.M.J., Villa, A.E.P. (eds.) ICANN 2017. LNCS, vol. 10614, pp. 64–71. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68612-7_8
34. Naguib, H., Narayan, N., Brügge, B., Helal, D.: Bug report assignee recommendation using activity profiles. In: Proceedings of the 10th Working Conference on Mining Software Repositories, pp. 22–30. IEEE Press (2013). https://doi.org/10.1109/MSR.2013.6623999
35. Xia, X., Lo, D., Wang, X., Zhou, B.: Accurate developer recommendation for bug resolution. In: 2013 20th Working Conference on Reverse Engineering (WCRE), pp. 72–81. IEEE (2013). https://doi.org/10.1109/WCRE.2013.6671282
36. Tian, Y., Wijedasa, D., Lo, D., Le Goues, C.: Learning to rank for bug report assignee recommendation. In: 2016 IEEE 24th International Conference on Program Comprehension (ICPC), pp. 1–10. IEEE (2016). https://doi.org/10.1109/ICPC.2016.7503715
37. Liu, C., Yang, J., Tan, L., Hafiz, M.: R2Fix: automatically generating bug fixes from bug reports. In: 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation, pp. 282–291. IEEE (2013). https://doi.org/10.1109/ICST.2013.24
38. Zhou, Y., Sharma, A.: Automated identification of security issues from commit messages and bug reports. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, pp. 914–919. ACM (2017). https://doi.org/10.1145/3106237.3117771
39. Yang, C.Z., Ao, C.M., Chung, Y.H.: Towards an improvement of bug report summarization using two-layer semantic information. IEICE Trans. Inf. Syst. **101**(7), 1743–1750 (2018). https://doi.org/10.1587/transinf.2017KBP0016
40. Li, X., Jiang, H., Liu, D., Ren, Z., Li, G.: Unsupervised deep bug report summarization. In: Proceedings of the 26th Conference on Program Comprehension, pp. 144–155. ACM (2018). https://doi.org/10.1145/3196321.3196326