

A Method of Component Prediction for Crash Bug Reports Using Component-Based Features and Machine Learning

Yang Xu* Chao Liu[†] Yong Li[‡] Qiaoluan Xie[‡] Hyun-Deok Choi[‡]
 SAP Labs China SAP Labs China SAP Labs China SAP Labs China SAP Labs Korea
 Xi'an, China Xi'an, China Xi'an, China Xi'an, China Seoul, South Korea
 yang.xu04@sap.com loch.liu@sap.com yong01.li@sap.com queena.xie@sap.com hyun.deok.choi@sap.com

Abstract—In large-scale software with hundreds of components, assigning a bug report to a relevant component representing a group of developers who can analyze and solve the potential issue can be a challenging task. In such a practical environment, we observed that bug reports often get re-assigned multiple times because of incorrect root cause component identification. Each re-assignment considerably delays the bug fix time. Therefore, it is desirable to identify a suitable component correctly. We report on our experience with component prediction when call stacks are used as the primary data of bug reports. Our method extracts the component features by aggregating the frequency of component functions from the call stack after introducing the component name and position into the original call stack. By predicting the component position of the call stack using component features and machine learning, we can obtain a suitable component of the bug report from the call stack. Our method achieves 73% accuracy on crash bug reports for SAP HANA, a large industrial in-memory database system. It significantly reduces the time delay for assigning bug reports to suitable components.

Index Terms—bug assignment, feature extraction, machine learning, crash failure, software evolution

I. INTRODUCTION

SAP HANA is an industrial in-memory database system that provides real-time data analysis capabilities for enterprises. A crash failure is a serious problem for a database system that can affect the operations of an enterprise business. Once we detect a new crash failure on SAP HANA in the internal delivery pipeline or the customer environment, it is common to open a new bug report for tracking and assign the bug to a component for fixing. Instead of assigning to individual developers, we assign the bug report to a relevant component because changes in developers (such as internal job transfers, demission, and new employee entry) are frequent, but the software components are relatively stable. The developer responsible for the component can then fix the bug. However, the existing method (Top-1 component method which we used the first component of the call stack as a bug component) cannot handle the case in which the root cause component is not in the first component of the call stack. It requires developers to reassign the bug report multiple times until the correct component is found, which is time-consuming. Fig. 1 shows the bug report assignment process.

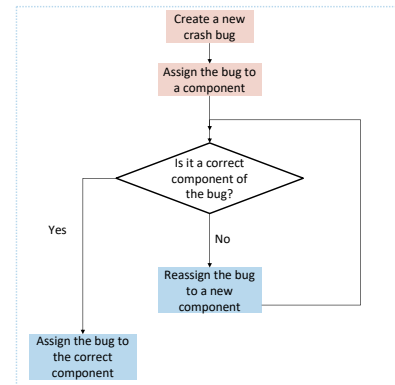


Fig. 1. Bug report assignment process

SAP HANA has hundreds of components, and getting the correct root cause component of an issue in these interactive components can take time and effort. In addition, components continue to grow as features and functionalities are added. Therefore, assigning bug reports to the correct components is a challenging task. There have been some methods for bug assignment [1], [2], [3], [4], which utilize the bug report information (e.g., bug description) as the primary data. However, we predict the component by extracting the component features, which aggregates the frequency of component functions from the call stack after introducing the component information into the original call stack. The call stack is a sequence of function calls in memory when an issue occurs and is probably better than the issue described in natural language in bug reports. Besides the bug assignment, due to the complexity of large-scale software, research on the improvement of bug triage in different aspects is promoted. For example, detecting duplicate bug reports [5], [6], [7] and evaluating the quality of the bug reports [8], [9].

This paper presents a method focused on component prediction using call stacks as the primary data, which predicts the component through several steps. Firstly, we preprocess original call stacks to introduce the component name and po-

sition by component mapping from the source code. Then, we extract the component features by aggregating the frequency of component functions from the call stack. Next, we build the best machine learning model and utilize the component position as the training label. By the predicted component position, we can obtain the component name from the call stack.

The contribution of this paper can be summarized as follows:

- We present a method to improve the bug assignment by extracting the component features using aggregating the frequency of component functions after introducing component information into the original call stack.
- We build a machine learning model based on the component features and utilize the component position instead of the component name as the label for model training.
- We obtain the component of the bug report from the call stack by the predicted component position from the machine learning model.

The remaining parts of this paper are organized as follows. We detail each step of the method in Section II. The evaluation result of the method is shown in Section III. Threats to validity are presented in Section IV. Section V and Section VI describe the related work and conclusion.

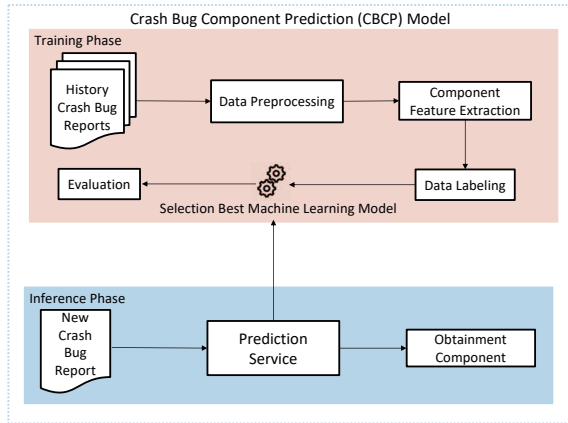


Fig. 2. Overall structure of the crash bug component prediction model

II. METHODOLOGY

A. Overview

Fig. 2 shows the overall structure of the component prediction model. The CBCP model includes the training phase and inference phase.

- In the training phase, firstly, this model collects crash bug reports from Bugzilla [10] as the dataset. Secondly, preprocess the original call stack from the dataset to remove irrelated information and add the component information. Then, calculate each component's feature in the call stack. Next, label the training dataset using the component position of the call stack and iterate various machine learning algorithms to choose a suitable

algorithm by evaluation. Lastly, pick the best machine learning model by tuning parameters.

- In the inference phase, the prediction service extracts the component features from the call stack of a new crash. Then, the best machine learning model predicts the component position using the component features. Finally, we obtain the component of the bug report using the predicted component position from the call stack.

B. Training Phase

1) *History Bug Reports Collection*: The method collects the historical crash bug reports from the Bugzilla repository and then extracts the original call stack and component from each bug report as the dataset. The original call stack consists of the function serial number, function name, Symbol, Sframe, Regs, and source file.

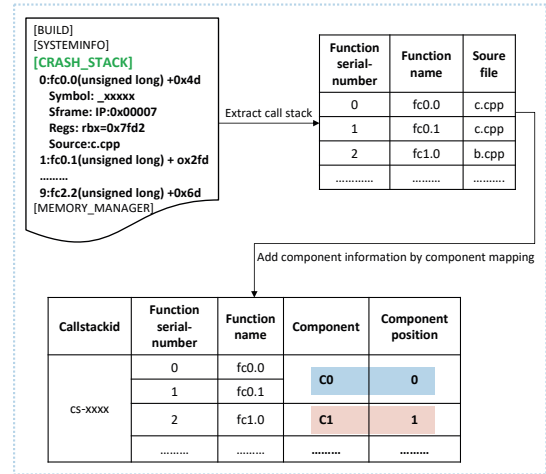


Fig. 3. The call stack preprocessing process

2) *Data Preprocessing*: There is a crash dump file for every crash failure in SAP HANA, which holds all kinds of information about the crash failure that occurs. Fig. 3 shows the call stack preprocessing process. Firstly, extract the [CRASH_STACK] part of the crash dump file as a call stack. Then, remove irrelated information (e.g., Symbol, Sframe) from the call stack and retain the function serial number, function name without parameter value, and source file. Next, add the component name into the call stack by component mapping. We extract the component mapping from the CMakeLists [11] of the source code, which includes the relationship between the component name and the source file. Each function in the call stack has a corresponding source file so that we can extract the corresponding component name of each function from the component mapping. By adding the component name, we can segment the call stack into blocks with functions. Finally, introduce the component position (from zero) into the call stack, which is the sequence number of each component in the call stack. In this study, we utilize the component position as the training label.

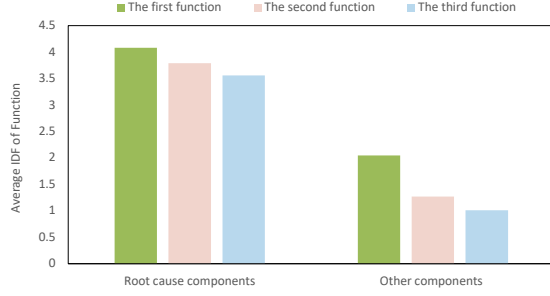


Fig. 4. The distribution of frequency of component functions

3) *Component Feature Extraction*: We observe that the frequency of functions of root cause components is lower than that of other components in our dataset through the statistical information in Fig. 4. A larger Inverse Document Frequency (IDF) shows a lower frequency for functions of call stacks. So the low-frequency functions of the component are unusual, which may indicate the root cause component from the call stack. The accuracy of obtaining the root cause component with the lowest frequency function is down than integrating the multiple low-frequency functions. Therefore, we fuse several functions and the position of each component to extract component features. And we find that the accuracy is similar using the first two functions of each component and more functions for extracting component features. So we choose the first two functions of each component to simplify the calculation process based on reasonable accuracy. To extract the component features, we first calculate the frequency of functions in the call stack by IDF based on the stack corpus of SAP HANA. Then we can extract component features by aggregating the IDF of component functions from the call stack. Fig. 5 is the component feature extraction process. There are four features for each component of the call stack:

- Feature1 is the component position.
- Feature2 is the IDF of the first function multiplied by the IDF of the second function.
- Feature3 is the IDF of the first function.
- Feature4 is the IDF of the second function.

Equation (1) is the IDF formula of the function in the call stack.

$$IDF(func) = \log \frac{Count(number\ of\ components)}{Count(number\ of\ function)} \quad (1)$$

- *func* is the function name of the call stack.
- *Count(number of components)* calculates how many call stacks have the components of the call stack in the stack corpus.
- *Count(number of function)* calculates how many call stacks have the function name in the stack corpus.

4) *Data Labeling*: We extract the call stack and bug component from each bug report as the dataset. The data label is the component position of the bug component (root cause component) in the call stack. For example, the label

The preprocessed call stack with idf of function

Callstackid	Component name	Component position	Function name	IDF of function
cs-xxxx	C0	0	fc0.0	3.1
			fc0.1	2.9
			fc0.2	1.8
	C1	1	fc1.0	2.1
			fc1.1	2.4
			fc1.2	1.7
	C2	2	fc2.0	1.8
			fc2.1	1.2
			fc2.2	1.7

Component feature of the preprocessed call stack

Callstackid	Component name	Feature1	Feature2	Feature3	Feature4
cs-xxxx	C0	Position of C0: 0	IDF(fc0.0)*IDF(fc0.1): 3.1*2.9 = 8.99	IDF(fc0.0): 3.1	IDF(fc0.1): 2.9
	C1	1	2.1*2.4 = 5.04	2.1	2.4
	C2	2	1.8*1.2 = 2.16	1.8	1.2

Fig. 5. The component feature extraction process

is two since the component position of bug component C2 is two in the call stack in Table I. The component position is more reliable than the component name as the training label in this study. About 96% of the root cause components are in the top five component positions of call stacks by statistics. More than 200 components in SAP HANA keep growing with new functionalities. If we utilize the component name as the label, once a component is not in the training labels, the Machine Learning (ML) model might predict an incorrect component. However, we can anticipate a suitable component because using the component position as the label can minimize the number of label categories and doesn't care about the component changes (e.g., new coming components).

TABLE I
DATA LABELING

The preprocessed call stack				Bug component (Root cause component)	Label
Callstackid	Function name	Component	Component position		
cs-xxxx	fc0.0	C0	0	C2	2
	fc0.1				
	fc0.2				
	fc1.0	C1	1		
	fc1.1				
	fc1.2				
	fc2.0	C2	2		
	fc2.1				
	fc2.2				

5) *Selection Best Machine Learning Model*: To choose a suitable machine learning algorithm, we iterate various machine learning algorithms, which include decision tree [12], random forest [13], and deep learning (MPL [14], LSTM [15]). The random forest algorithm achieves the best accuracy by performance evaluation. Fig. 6 is the iteration ML algorithms process. Furthermore, we tune the parameters of the random forest algorithm by GridSearchCV [16]. Then we can select the best machine learning model from the random forest algorithm according to the best parameter values from tuning.

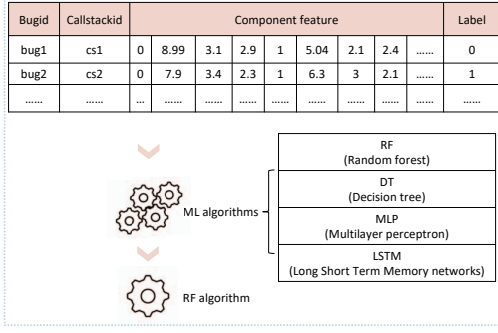


Fig. 6. Iteration Machine Learning algorithms process

C. Inference Phase

1) *Prediction Service*: Firstly, preprocess the original call stack from the new crash bug report to add component information. Secondly, extract the component features from the preprocessed call stack. Finally, the best ML model predicts the component position based on the component features. Fig. 7 is the workflow of the prediction service.

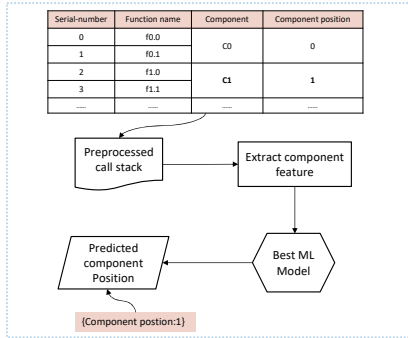


Fig. 7. Workflow of prediction service

2) *Obtainment Component*: Our purpose is to obtain the component of the bug report from the call stack. Previously step, we get the predicted component position from the prediction service. Moreover, the component name and position are paired in the preprocessed call stack. So, we can obtain the component name from the call stack through the predicted component position. For example, the component of the bug report for the predicted component position one is C1 in Fig. 7.

III. EVALUATION

A. Data Set

To evaluate our method, we collected 10618 crash bug reports between 2012/10/08 and 2022/10/10. These bug reports come from internal testing and external customers with different code branches of SAP HANA.

B. Performance Measure

Accuracy is widely used to examine model performance. So, we use the accuracy rate to evaluate the performance of

our method. We assume that the total number of bug reports is *Total number of bug reprot*s. The total number of bug reports where our method predicts the correct component is *Bug component assigned correctly*, then the accuracy rate can be computed as follows.

$$Accuracy = \frac{Bug\ component\ assigned\ correctly}{Total\ number\ of\ bug\ reprot\ s} \quad (2)$$

C. Result and Discussion

Cross-validation [17] is a statistical method for evaluating model performance. In cross-validation, the data is divided several times and combined into different training datasets and test datasets. It is more stable and comprehensive than dividing the training and test datasets set once. We used 10-fold cross-validation for performance evaluation, and the result is in Table II. We can be sure our method can get better results by evaluation. We also verified 179 new crash bugs from customer environments and achieved 71.321% accuracy. From the experimental results, we find that this method has obvious improvement in the case that the call stack contains basic components (e.g., basis, memory management).

Currently, we are using the component mapping of the master code branch to introduce component information into the call stack. In this way, we cannot accurately find component information for the call stack of other code branches, which affects our accuracy. To improve accuracy, we need to generate different component mapping for different code branches. In addition, we found that some components have no relation to the root cause of an issue, which we called “stop components”. It can also improve the accuracy after deleting the “stop components” in these call stacks.

TABLE II
ACCURACY COMPARISON USING 10-FOLD CROSS-VALIDATION

N-fold	CBCP model	Top-1 method	BR Stack Traces with categorical features [18]
1	72.638%	45.472%	63.561%
2	71.275%	47.809%	65.925%
3	72.736%	48.880%	62.022%
4	74.878%	47.711%	62.888%
5	74.196%	50.243%	63.996%
6	74.099%	48.003%	61.876%
7	72.833%	47.224%	63.314%
8	72.514%	49.220%	63.902%
9	72.807%	47.855%	61.461%
10	75.536%	51.851%	62.923%
Mean-accuracy	73.351%	48.426%	63.186%

IV. THREATS TO VALIDITY

We identify the following threats to validity:

- *Imbalanced data*: In the multi-classification model, we find that different label categories correspond to varying sizes of datasets, which impacts model learning. Therefore, we will apply some balancing techniques to improve performance.

- **Restriction:** This method cannot handle the root cause component that doesn't arise in the call stack. For example, the root cause component of a crash caused by infrastructure usually doesn't appear in the call stack.

V. RELATED WORK

Sabor et al. [18] present a method that combines the stack traces and categorical features similarity to predict the component by KNearest Neighbor (KNN). They convert the call stack to a feature vector for calculation similarity between the new bug and history bug reports. However, we rely on the call stack to extract component features rather than calculate the similarity of call stacks. Yan et al. [19] propose a component recommendation algorithm that uses the Latent Semantic analysis (DPLSA) model and Jensen-Shannon divergence (DPLSAJS). The method is based on discriminant probability and extracts the feature from the bug description. Chen et al. [20] present an approach utilizing the bug-tossing history to build a bug-tossing graph. Then calculates the similarities between new and existing bugs to choose the developer for fixing issues. Zhang and Challis [21] present the utilization of the multi-layer neural network to build component classifiers for predicting the component of bug reports. The method extracts the features from bug descriptions. Cubranic and Murphy [22] propose applying Bayesian machine learning techniques to predict the developer using the text categorization from the bug report's description. Anvik et al. [23] propose applying the Support Vector Machine (SVM) algorithm to predict the developer for fixing a bug. The method learns the kinds of bug reports from summary and description. The above methods utilize the bug report information (e.g., bug description, bug summary, bug tossing history) as the main data to predict the proper developer or component of new coming bugs. However, we utilize the call stack with component information to predict the suitable component of the bug report.

VI. CONCLUSION

This paper proposes a method to predict the component of the bug report after introducing component information into the call stack. We extract the component features based on the IDF of component functions in the call stack because low-frequency functions of the component are unusual in the stack corpus, which may indicate the root cause component. Then, we build the machine learning classification model and utilize the component position as the training label. By the predicted component position, we can obtain the component from the call stack. The evaluation result shows our method can detect suitable components for bug reports.

In the future, we will try more advanced machine learning algorithms to improve accuracy. And the model training will be executed regularly and automatically without human effort.

VII. ACKNOWLEDGMENT

We thank Sascha Schwedes, Thomas Bach, and Dong-Won Hwang from SAP Labs for their engineering support. We also thank the anonymous reviewers for their feedback.

REFERENCES

- [1] J. Anvik, "Automating bug report assignment," in *Proceedings of the 28th international conference on Software engineering*, 2006, pp. 937-940.
- [2] P. Bhattacharya, I. Neamtiu, and C. R. Shelton, "Automated, highly-accurate, bug assignment using machine learning and tossing graphs," *Journal of Systems and Software*, vol. 85, no. 10, pp. 2275-2292, 2012.
- [3] M. Alenezi, K. I. Magel, and S. Banitaan, "Efficient Bug Triaging Using Text Mining," *J. Softw.*, vol. 8, pp. 2185-2190, 2013.
- [4] A. S. Badashian and E. Stroulia, "Vocabulary and time based bug-assignment: A recommender system for open-source projects," *Software: Practice and Experience*, vol. 50, pp. 1539 - 1564, 2020.
- [5] P. Runeson, M. Alexandersson and O. Nyholm, "Detection of duplicate defect reports using natural language processing," *Proceedings of the 29th international conference on Software Engineering*, pp. 499-510, 2007.
- [6] Y. Dang, R. Wu, H. Zhang, D. Zhang and P. Nobel, "ReBucket: A method for clustering duplicate crash reports based on call stack similarity," *2012 34th International Conference on Software Engineering (ICSE)*, 2012, pp. 1084-1093.
- [7] E. Su and S. Joshi, "Poster: Leveraging Product Relationships to Generate Candidate Bugs for Duplicate Bug Prediction," *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*, 2018, pp. 210-211.
- [8] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schroter and C. Weiss, "What Makes a Good Bug Report?," in *IEEE Transactions on Software Engineering*, vol. 36, no. 5, pp. 618-643, Sept.-Oct. 2010.
- [9] S. Huang, L. Chen, Z. Hui, J. Liu, S. Yang and Q. Chen, "A Method of Bug Report Quality Detection Based on Vector Space Model," *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, 2019, pp. 510-511.
- [10] N. Serrano and I. Ciordia, "Bugzilla, itracker and other bug trackers," *IEEE Software*, vol. 22, no. 2, pp. 11-13, 2005.
- [11] K. Martin and B. Hoffman, "Mastering CMake: A Cross-Platform Build System," Clifton Park, NY, USA: Kitware, 2010.
- [12] P. H. Swain and H. Hauska, "The decision tree classifier: Design and potential," in *IEEE Transactions on Geoscience Electronics*, vol. 15, no. 3, pp. 142-147, July 1977.
- [13] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5-32, 2001.
- [14] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," in *Neural networks*, vol. 2, no. 5, Elsevier, 1989, pp. 359-366.
- [15] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735-1780, 1997.
- [16] G. Ranjan, A. K. Verma, and S. Radhika, "K-nearest neighbors and grid search cv based real time fault monitoring system for industries," in *2019 IEEE 5th international conference for convergence in technology (I2CT)*, 2019: IEEE, pp. 1-5.
- [17] S. Yadav and S. Shukla, "Analysis of k-fold cross-validation over hold-out validation on colossal datasets for quality classification," in *2016 IEEE 6th International conference on advanced computing (IACC)*, 2016: IEEE, pp. 78-83.
- [18] K. K. Sabor, A. Hamou-Lhadj, A. Trabelsi, and J. Hassine, "Predicting bug report fields using stack traces and categorical attributes," in *Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering*, 2019, pp. 224-233.
- [19] M. Yan, X. Zhang, D. Yang, L. Xu, and J. D. Kymer, "A component recommender for bug reports using discriminative probability latent semantic analysis," *Information and Software Technology*, vol. 73, pp. 37-51, 2016.
- [20] L. Chen, X. Wang and C. Liu, "Improving Bug Assignment with Bug Tossing Graphs and Bug Similarities," *2010 International Conference on Biomedical Engineering and Computer Science*, 2010, pp. 1-5.
- [21] W. Zhang and C. Challis, "Software component prediction for bug reports," in *Asian conference on machine learning*, 2019: PMLR, pp. 806-821.
- [22] D. Cubranic and G. C. Murphy, "Automatic bug triage using text categorization," in *SEKE'04*, 2004, pp. 92-97.
- [23] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?," in *ICSE '06: Proceedings of the 28th international conference on Software engineering*. New York, NY, USA: ACM, 2006, pp. 361-370.