SCIENCE CHINA
Information Sciences

• LETTER •

May 2024, Vol. 67, Iss. 5, 159101:1–159101:2
https://doi.org/10.1007/s11432-023-3975-6

Improving actionable warning identification via the refined warning-inducing context representation

Xiuting GE, Chunrong FANG*, Xuanye LI, Quanjun ZHANG, Jia LIU*, Zhihong ZHAO & Zhenyu CHEN

State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093*, China*

Received 6 April 2023/Revised 15 June 2023/Accepted 9 November 2023/Published online 24 April 2024

Static analysis tools (SATs) have shown potential ability in software quality assurance [1]. However, SATs report an overwhelming number of warnings, where most warnings are unactionable ones (i.e., warnings that are not acted on or fixed by developers). It is not a trivial task for developers to find actionable warnings (i.e., warnings that are acted on and fixed by developers) in all reported warnings. Hence, the excessive unactionable warnings seriously impede the usability of SATs [2].

To improve the usability of SATs, many machine learning-based actionable warning identification (ML-based AWI) approaches are proposed and studied [3]. The general procedure of these approaches is to perform the warning representation, train an ML-based classifier based on the warning representation, and apply this classifier to classify warnings into actionable and unactionable ones. For these ML-based AWI approaches, one of the most critical parts is the warning representation, which is closely related to AWI performance. Currently, the hand-engineered, token-based, and text-based techniques are commonly used for the warning representation, which mainly focus on capturing the statistical or lexical information for AWI. However, such techniques miss structural information of the warning for AWI. The warning-inducing context, which is mainly extracted from the class/method containing the warning or program slicing for the warning, is the foundation for capturing structural information of the warning. Yet, due to the irrelevant and incomplete warning-inducing context caused by the existing extraction ways, it is challenging to capture structural information of the warning.

Our approach. We propose an AWI approach via the refined warning-inducing context representation, which aims to perform the AWI by capturing both lexical and structural information from the refined warning-inducing context. To obtain the refined warning-inducing context, our approach defines a new warning-slicing criterion for program slicing to construct the warning-inducing context and designs an adjustment algorithm to complete the warning-inducing context. Such a refined warning-inducing context can show the complete dependencies of the reported warning while removing the irrelevant information of this reported warning. Besides, our approach performs the AWI-aware abstraction for the refined warning-inducing context to help train a more generalizable AWI classifier. Subsequently, our approach leverages an advanced source code representation technique ASTNN [4] to learn the warning-inducing context representation with both lexical and structural information. Based on the representation of labeled warnings, our approach finally trains an ML-based classifier for AWI. Figure 1 shows the overview of our approach, which mainly contains the training phase and detection phase.

(1) The training phase. In the training phase, our approach includes five steps, i.e., the warning-inducing context construction, completion, abstraction, representation, and ML-based AWI classifier training.

Step 1. Given a warning, the natural way to obtain the warning-inducing context is to extract the source code from the class/method information containing a warning or the warning line numbers. However, the above way either brings much information that is irrelevant to the warning or is too coarse-grained due to missing the detailed warning context information. Thus, based on the warning line numbers, our approach leverages the program dependency analysis of Joana to perform the backward program slicing for the warning. In particular, our approach defines a new warning-slicing criterion for Joana, which aims to alleviate the unsolvable and time-consuming problem in the original criterion of Joana [5]. Such a new warning slicing criterion can be seen in Appendix A. In the end, our approach obtains the warning-inducing context, which can contain statement dependencies that report a warning while eliminating the irrelevant information of this warning.

Step 2. Due to the arbitrary and irregular coding format in the source code, the constructed warning-inducing context could be incomplete. To address the above problem, our approach designs an adjustment algorithm for the warning-inducing context completion. The core idea of our proposed adjustment algorithm is relying on abstraction syntax tree rules of the source code to extract statements with the corresponding source code line numbers from Joana, thereby obtaining the refined warning-inducing context. Appendix B shows the detailed completion process.

Step 3. Some specific identifiers (e.g., numbers, string literals, and project-specific variables) in the refined warning-inducing context provide little structural information to judge whether a warning is actionable or unaction-

* Corresponding author (email: fangchunrong@nju.edu.cn, liujia@nju.edu.cn)

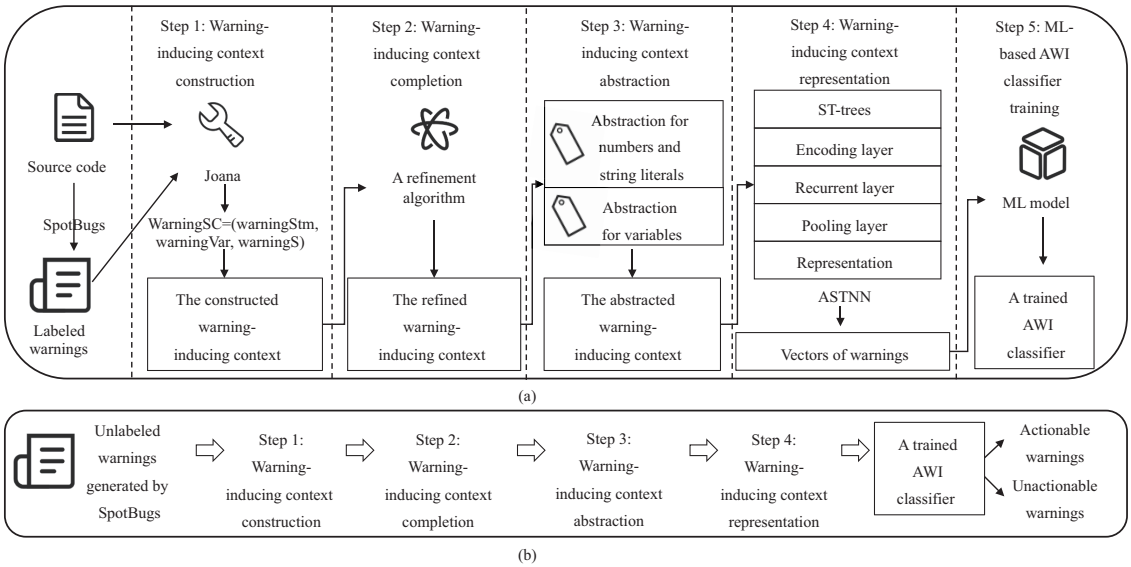

Figure 1 Overview of our approach. (a) Training phase; (b) detection phase.

able. Thus, our approach performs the AWI-aware abstraction for the warning-inducing context. On the one hand, our approach abstracts numbers/string literals into NUM/STR. On the other hand, our approach abstracts rare variables with UNK based on the frequency of variables.

Step 4. Given the abstracted warning-inducing context with a set of statements, our approach relies on ASTNN to split them into a sequence of statement trees, learn vector representations of multi-way statement trees, and track the naturalness among statement trees. To the end, our approach outputs an m-dimension vector, which is the warning representation with lexical and structural information.

Step 5. Our approach uses an ML model to train an AWI classifier based on the vectors of labeled warnings. Then, this classifier is used for the detection phase.

(2) The detection phase. Our approach relies on Steps 1–4 in the training phase to obtain the vectors of unlabeled warnings. Based on these vectors, our approach uses the AWI classifier from the training phase to classify the unlabeled warnings into actionable and unactionable ones.

Evaluation. To validate our approach, we conduct the experimental evaluation on 51K+ warnings, which are collected from 56 releases of five large-scale and open-source projects. We compare our approaches with four state-of-the-art ML-based AWI approaches (i.e., one hand-engineered AWI approach, one token-based AWI approach, and two text-based AWI approaches) in the within-project and cross-project AWI. The experimental results show that in the within-project AWI, our approach improves four approaches by 4%–38% in terms of AUC. In particular, our approach achieves the best AUC (nearly 100%) in some cases. In the cross-project AWI, our approach improves four approaches by 5%–21% in terms of AUC. Through the Scott-Knott test, the results show that compared with four state-of-the-art approaches, our approach is ranked first in the within-project and cross-project AWI.

Conclusion. We improve AWI via the refined warning-inducing context representation, which captures both lexical and structural information for AWI from the refined warning-inducing context. We conduct experiments on over 51K+ warnings from 56 releases of five large-scale and open-source projects. The results in both within-project and cross-project AWI show that our approach is more effective than four state-of-the-art ML-based AWI approaches.

Acknowledgements The work was partly supported by National Natural Science Foundation of China (Grant Nos. 61932012, 62141215, 62272220, 62372228) and Science, Technology and Innovation Commission of Shenzhen Municipality (Grant No. CJGJZD20200617103001003).

Supporting information Appendixes A and B. The supporting information is available online at info.scichina.com and link.springer.com. The supporting materials are published as submitted, without typesetting or editing. The responsibility for scientific accuracy and content remains entirely with the authors.

References

1 Smith J, Johnson B, Murphy-Hill E, et al. How developers diagnose potential security vulnerabilities with a static analysis tool. IEEE Trans Software Eng, 2019, 45: 877–897
2 Wang J, Wang S, Wang Q. Is there a "Golden" feature set for static warning identification? An experimental evaluation. In: Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, 2018. 1–10
3 Muske T, Serebrenik A. Survey of approaches for postprocessing of static analysis alarms. ACM Comput Surv, 2022, 55: 1–39
4 Zhang J, Wang X, Zhang H, et al. A novel neural source code representation based on abstract syntax tree. In: Proceedings of the 41st IEEE/ACM International Conference on Software Engineering, 2019. 783–794
5 Weiser M. Program slicing. IEEE Trans Software Eng, 1984, SE-10: 352–357