Adaptive Probabilistic Model for Ranking Code-Based Static Analysis Alerts

Sarah Smith Heckman
North Carolina State University, Campus Box 8206, Raleigh, NC 27695
sarah_heckman@ncsu.edu

Abstract

Software engineers tend to repeat mistakes when developing software. Automated static analysis tools can detect some of these mistakes early in the software process. However, these tools tend to generate a significant number of false positive alerts. Due to the need for manual inspection of alerts, the high number of false positives may make an automated static analysis tool too costly to use. In this research, we propose to rank alerts generated from automated static analysis tools via an adaptive model that predicts the probability an alert is a true fault in a system. The model adapts based upon a history of the actions the software engineer has taken to either filter false positive alerts or fix true faults. We hypothesize that by providing this adaptive ranking, software engineers will be more likely to act upon highly ranked alerts until the probability that remaining alerts are true positives falls below a subjective threshold.

1. Introduction

Software engineers tend to repeat mistakes that lead to software faults[1] [2]. Automated static analysis (ASA) tools can find these mistakes and provide valuable feedback about the correctness of code [2] early in the development life cycle. However, ASA tools report on average 66% false positives (FP) [1, 2, 4]. Due to the need for manual inspection of alerts [4], a high number of FPs may make an ASA tool too costly to use.

The objective of this research is *to mitigate the cost of false positives by developing an adaptive probabilistic model to rank alerts generated from automated static analysis tools by the probability that an alert is a true fault in the system.* We will investigate the following theories: 1) the probabilistic model with rank true positive (TP) alerts with a higher probability than FP alerts and 2) software engineers will continue to investigate alerts until the probability that the remaining alerts are TPs falls below the software engineer's subjective threshold. The Automated Warning Application for Reliability Engineering[2] (AWARE) tool automates the adaptive ranking model and gathers data to validate the proposed theories.

2. Proposed solution

The focus of this research is on the development of a probabilistic model for the adaptive ranking of alerts generated by ASA tools to mitigate the cost of FPs.

2.1. Key model concepts

Software engineers will select an alert from the ranked list and inspect the associated source code for the possible fault. Filtering or closing an alert removes the alert from current or future rankings:

- A software engineer *filters* an alert when no fault is observed in the specified source code.
- The alert is *closed* when an ASA tool no longer identifies an alert. An alert is closed by an alert fix or configuration change of the ASA tool.

2.2. Alert ranking model

ASA alerts are ranked by the probability that an alert is a TP in the system. In the current version of the model, AWARE v0.3, three factors contribute to the ranking of an ASA alert: Type Accuracy (TA), Code Locality (CL), and Generated Test Failure (GTF). The total probability is a weighted combination of TA and CL (where weights are represented by the β_{TA} and β_{CL} coefficients) when GTF is not 1. If GTF is 1, then a concrete test case was generated that causes a failure. Equation 1 describes how the probability (P) that a single alert (α) is a TP is calculated.

$$\mathrm{P}(\alpha) = \begin{cases} \beta_{TA}(TA(\alpha)) + \beta_{CL}(CL(\alpha)), GTF \neq 1 \\ 1, GTF = 1 \end{cases} \qquad (1)$$
$$where\ \beta_{TA} + \beta_{CL} = 1$$

The number of filtered and closed alerts in a population of alerts adjusts the TA and CL values via

[1] For this research, *fault* specifies a manifestation of a mistake. A fault may never surface as a failure during operational use.

[2] Download AWARE from: http://agile.csc.ncsu.edu/aware.

the adjustment factor (AF) described in Equation 2. A population (p) of alerts is a subset of alerts from the total population that share some characteristic (e.g. the same alert type or location).

$$AF_p = 1.0 - (\#filtered_p / total_p) + (\#closed_p / total_p) \quad (2)$$

Type accuracy, described in Equation 3, is the probability that an alert is a TP based upon its type (e.g. cast error, null pointer, etc.). ASA tools have different ratios of TPs to FPs dependent on the individual checker and the code under analysis [2]. The TA value of alert (α) is adjusted based on software engineer feedback on the population of alerts of the same alert type as α. Each alert type is seeded with an initial TA value (τ), which reflects the ratio of TPs to all alerts found by the ASA checker for the type. τ may be based on literature or historical, empirical evidence.

$$TA(\alpha) = \tau \cdot AF_{type} \quad \mathbf{(3)}$$

Code locality, described in Equation 4, is the probability that an alert is a TP based on the alert's location in the source code. The CL value of alert (α) is adjusted based on the number of filtered and closed alerts in the same method, class, or package of α [3]. The contribution of each code location to the CL value is weighted based on literature or historical, empirical evidence and represented by a β coefficient.

$$CL(\alpha) = \beta_{method}(AF_{method}) + \beta_{class}(AF_{class}) + \beta_{package}(AF_{package}) \quad where\, \beta_{method} + \beta_{class} + \beta_{package} = 1 \quad (4)$$

Generated test failure, described in Equation 5, is the probability that an alert is a TP based on the failure of test case(s) generated from the alert's feedback [1]. A failing test cases describes an exercisable error, therefore the probability of the alert being a TP is 1.

$$GTF(\alpha) = \begin{cases} 1, & if\ any\ test\ fails \\ 0, & if\ all\ tests\ pass \end{cases} \quad (5)$$

3. Research methodology and evaluation

The research methodology will consist of developing and evolving the probabilistic model to rank ASA alerts via a literature search, intellectual work, and formative cases studies. Over the course of the research, the contribution of each factor will be ascertained, and factors may be modified as necessary. Other factors will be investigated for inclusion in the final probabilistic model. The best ranking factors for the model will be determined by looking at several versions of active open source projects written in Java with associated bug databases. A subset of projects will calibrate the probabilistic model that is used to predict the ranking of ASA alerts for the remaining projects. Correlations between factors and the strength of the contributions of each factor will be measured. The factors with the strongest contribution will become part of the ranking model.

Finally, empirical case studies will be conducted with industrial partners. Model calibration will occur via historical studies of the project under analysis. Data about the model will be gathered in process by AWARE.

4. Summary

The expected contribution of this research is to mitigate the cost of FPs by developing an adaptive probabilistic model to rank alerts generated from ASA tool(s) by the probability that an alert is a TP in the system. Feedback from software engineers in the form of filtering alerts found to be FPs and fixing alerts found to be TPs modifies the ranking of remaining ASA alerts. Ranking alerts could make the use of ASA tools more tractable by reducing the cost of FP alerts. While this research focuses on the Java™ programming language, the generation of a probabilistic model for other languages should follow.

5. Acknowledgements

This work is supported in part by the Center for Advanced Computing and Communication grant and an IBM PhD Fellowship Award. I would like to thank the RealSearch reading group for their feedback.

6. References

[1] C. Csallner and Y. Smaragdakis, "Check 'n' Crash: Combining Static Checking and Testing," in *27th International Conference on Software Engineering*, St. Louis, MO, USA, 2005, pp. 422-431.

[2] D. Hovemeyer and W. Pugh, "Finding Bugs is Easy," in *19th ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications*, Vancouver, British Columbia, Canada, 2004, pp. 132-136.

[3] T. Kremenek, K. Ashcraft, J. Yang, and D. Engler, "Correlation Exploitation in Error Ranking," in *12th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, Newport Beach, CA, USA, 2004, pp. 83-93.

[4] S. Wagner, J. Jrjens, C. Koller, and P. Trischberger, "Comparing Bug Finding Tools with Reviews and Tests," in *17th International Conference on Testing of Communicating Systems*, 2005, pp. 40-55.