Automated Adaptive Ranking and Filtering of Static Analysis Alerts

Sarah Heckman and Laurie Williams
North Carolina State University
{sarah_heckman, lawilli3}@ncsu.edu

Abstract

Static analysis tools are useful in finding recurring software faults and weaknesses during the development process. However, these tools often report a high number of false positives, dissuading software engineers from frequent use of the tools during development. By ranking static analysis alerts by the probability the alert is a true positive, software engineers can be directed to the faults that are most likely to need attention. The ranking is based on historical data from the filtering of alerts previously found to be false positives by a software engineer. The Automated Warning Application for Reliability Engineering (AWARE) v 0.2 has been created to support static analysis alert ranking and filtering. Initial results from a feasibility study show that with AWARE, true positive alerts appear at the top of the ranking and the distance between true positive alerts are better than a random ordering of alerts. By filtering a small number of false positives, AWARE can provide true positive alerts to the software engineer.

1. Introduction

Software engineers tend to repeatedly make the same mistakes [3] leading to software faults[1]. Static analysis, the process of evaluating a system or component based on its form, structure, content, or documentation [4] without execution of the source code, is useful for detecting these recurring faults. Automated static analysis tools provide invaluable information about the correctness and security of code; however, these tools report a high number of false positives (FP) [1, 3, 6].

The objective of our research is *to improve the correctness and security of a system by continuously, automatically, and efficiently providing adaptively-ranked and filtered static analysis alerts.* For purposes of this research an alert is defined as a notification to a software engineer, of a potential fault in the source code that has been identified via static analysis. An Eclipse[2] plug-in has been created to display and rank alerts generated from static analysis. This tool is called the **A**utomated **W**arning **A**pplication for **R**eliability **E**ngineering[3] (AWARE).

To mitigate the high number of reported FPs alerts are ranked in AWARE by the probability that an alert is a true fault, or true positive (TP), in a system. Alerts found to be FPs can then be filtered by the software engineer. After an alert is filtered, the ranking of all remaining alerts is adapted based on this feedback. Alerts found to be TPs are corrected by the software engineer. Subsequent runs of a static analysis tool will no longer report the alert, and the alert will be closed as a TP. Alerts generated in future runs of a static analysis tool will be ranked based on filtered and closed alerts. The initial ranking of AWARE is compared with a random baseline in a feasibility study.

2. Alert Ranking and Filtering

Alert ranking is the practice of ordering static analysis alerts such that alerts more likely to be true faults are listed first [5]. An adaptive ranking scheme means that if an alert is found to be a TP or FP, that information is used to modify the rankings of other alerts in the system [5]. Software engineer feedback, in the form of 1) filtering alerts found to be FPs [7]; and 2) from fixing (closing) alerts, is used to adjust the ranking of non-filtered, non-closed alerts and alerts detected in future runs of the system. AWARE v 0.2 uses data from filtered alerts to calculate the probability that an alert is a TP, but future versions will also include fixed alerts in the calculation.

Below are the factors used in the current ranking algorithm in AWARE v. 0.2. Factors will be added, removed, and revised as research progresses.

- **Type accuracy (TA):** the probability that a particular type of alert (e.g. a cast error exception), generated by a static analysis tool is a TP.

[1] We use the term fault to specify a manifestation of a mistake, or programmer error. Potentially, a fault may never surface as a failure during operational use of a product.

[2] Eclipse is an open source development environment. Functionality of Eclipse is extended via plug-ins.

[3] AWARE can be found at http://agile.csc.ncsu.edu/aware

- **Code locality (CL):** the probability that an alert is a TP given that other alerts in the same population are also TPs. A population is a particular area of code, like a function, class, or package [5].
- **Generated test failure (GTF):** the probability an alert is a TP if any of the associated automatically-generated test cases fail.

We expect that the contribution of each factor will vary by project and/or programmer. Therefore, a regression equation will be used with the probability that an alert is a TP as the dependent variable. The importance of each factor to the regression equation will be calibrated, similar to [2]. Additionally, the ranking metric will be revised to examine risk exposure for each alert. Risk exposure is defined as probability that an alert is a TP * severity of the alert.

3. Feasibility Study

AWARE automatically and continuously provides the software engineer with a listing of static analysis alerts ranked by the probability that an alert is a TP in the system. Currently, alerts are reported from the static analysis tool Check 'n' Crash [1] (CnC). CnC combines static analysis with automatic concrete test case generation to reduce the number of FPs.

AWARE was run on the RealEstate[4] example, generating 28 alerts. RealEstate is a Java program that consists of 775 lines of code and was written by the first author and another graduate student. Of the 28 alerts, only 27 were analyzed because two alerts of the same type were on the same line. These alerts are considered the same by AWARE v0.2, and were combined for the experiment. Two of the 27 alerts were found to be TPs.

This experiment investigated two questions: 1) Does AWARE's initial ranking strategy rank TP alerts higher in the listing than a random ordering of alerts would rank TP alerts? and 2) What percentage of FPs must be filtered before all of the TPs reach the top of the ranking? The first question was investigated via the following metrics: the number of initial FPs before the first TP (M1), and the average number of FPs between TPs (M2). Fifty random permutations where obtained and the average was taken across the fifty samples for each metric. For M1, AWARE returned a TP at the top of the list. There was an average of 8.9 FP alerts before the first TP alert in the random samples. For M2, AWARE had an average of four FPs between TPs. For random, there was an average of 8.1 FPs between TPs.

The second question was answered using a single metric: the number of alerts that were filtered before all TP alerts reached the top of the ranking at least once. In the feasibility study there were three different sets of initial TA values for each CnC alert type. For one set of the initial TA values, six alerts (over a quarter of reported alerts) were required to be filtered before all TPs reached the top of the ranking. In the other two cases three alerts (11% of the reported alerts) required filtering. While the results are not statistically significant, they are encouraging that AWARE's static analysis ranking scheme is viable for presenting software engineers with alerts more likely to be TPs.

4. Conclusions

This research investigates ranking alerts reported by static analysis tools according to the probability that an alert is a true fault in a system. The ranking system is adaptive by incorporating programmer feedback in the form of filtering alerts found to be FPs. The goal of the proposed alert ranking scheme is to reduce the overall time for a fault fix and improve code quality by providing software engineers feedback about the correctness and security of their code early and often.

Acknowledgements

This work is supported by the NCSU Center for Advanced Computing and Communication and an IBM PhD Fellowship.

References

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

[2] M. Davidsson, J. Zheng, N. Nagappan, L. Williams, and M. Vouk, "GERT: An Emperical Reliability Estimation and Testing Feedback Tool," 15th IEEE International Symposium on Software Reliability Engineering, St. Malo, France, 2004, pp. 269-280.

[3] D. Hovemeyer and W. Pugh, "Finding Bugs is Easy," Conference on Object Oriented Programming Systems Languages and Applications (OOSPLA) Companion, Vancouver, BC, 2004, pp. 132-135.

[4] IEEE, "IEEE Standard Glossary of Software Engineering Terminology," IEEE Standard 610.12-1990.

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

[6] N. Rutar, C. B. Almazan, and J. S. Foster, "A Comparison of Bug Finding Tools for Java," 15th IEEE International Symposium on Software Reliability Engineering, St.Malo, Bretagne, France, 2004, pp. 245-256.

[7] S. E. Smith, L. Williams, and J. Xu, "Expediting Programmer AWAREness of Anomalous Code," 16th IEEE International Symposium on Software Reliability Engineering, Fast Abstract, Chicago, IL, 2005

[4] http://open.ncsu.edu/se/realestate