A Systematic Literature Review on Action Alert Identification Strategies for the Analysis of Automated Static Code

1. Introduction

Static analysis refers to the process which involves evaluation of a component or system based on its structure, form, documentation or content (Myers, 2009). Automated Static Analysis (ASA) can determine common problems in coding early on in the development process, using a tool which automates source code inspection. ASA then reports possible anomalies in the source code, often called alerts, which come in the form of buffer overflows, null pointer dereferences, as well as style inconsistencies. Developers will then work towards inspecting every alert in order to identify whether or not an alert is indeed an indication of a viable anomaly which requires to be fixed (Stan & Fowler, 2011). If a developer indeed determines that the alert is valid and fixable, it becomes an 'actionable alert'. When the alert does not prove to be an anomaly, or if it is viewed as unimportant to the developer, a source code anomaly which is inconsequential to the functionality of the program as perceived by the developer, then the alert is termed as an 'unactionable alert' (Harris, 2012).

2. Overview of the systematic literature review method

We used the described SLR guidelines by Kichel (2008) in order to develop our protocol. This protocol is used in addressing the different research objectives as proposed in the study. It describes the question, research strategy for searching for relevant studies, selected studies analysis, as well as data synthesis.

2.1 Research Questions

We have derived our questions used in the research directly from the list of SLR objectives. We want to answer the following criteria:

- What are the different categories of artifacts that are used for AAIT input?
- What are the approaches used for the AAIT?
- What conclusions can we get regarding the efficacy of AAITs from the results gathered in the chosen studies?
- What are the challenges encountered during research?

Since AAITs are done after ASA, we are interested in first understanding the information sources used in generating the prioritization or classification of an alert. Afterwards, we want to determine the underlying algorithms involved in prioritizing or classifying alerts (Simmon, 2010).

2.2 Search strategy

This section covers the process involved in generating search strategy, terms, searched databases, and the documentation used in the search.

2.3 Search strategy and terms

We have identified some key terms which were used for the search from previous experience in the subject area. The main term used for the search is ‘static analysis’ in focusing on solutions which determine actionable alerts when performing ASA (Roldenson & Waltz, 2003). The other search terms are classified into two: techniques for identification and descriptive alert names generated by static analysis.

3. Overview of Studies

We have identified 23 studies in the literature which focus on prioritizing or classifying alerts that are generated by the ASA. A quick look at the studies show that, all work performed on AAITs were done during or after the year 2003, except one, and most of them were published in 2007 to 2008. (Walter, 2010). On top of that, we have also considered the publication venues for the papers selected.

4. Software Characteristics

One common characteristic among AAITs is that they use additional information regarding software artifacts with the purpose of prioritizing or classifying alerts as either actionable or unactionable. This additional information is called the software artifact characteristics, serving as an independent variable when it comes to predicting the so-called actionable alerts (Mosley, Beuby, & Walter, 2008).

5. Classification AAITs

These classification AAITs divide the alerts into two batches: the alerts which are likely to be actionable, as well as alerts which are most likely to become unactionable. (Gosby, 2010). For every AAIT, we reported in the paper showing the description of the AAIT, the input in the form of used artifact characteristics, the ASA used, AAIT type, programming language used, as well as the research methodology. If there is no name used in the selected study, we make a name according to the first letter of the last names of the first three authors, as well as the last two numbers of the publication year. (Moffat, 2010).

References

Gosby, H.A. (2010). Integrating dynamic and static analysis for the detection of vulnerabilities. In: The 30th Annual Global Computer Application Software, Chicago, Illinois, USA. August 16 – 20, 2010, pp. 34-56.

Harris, J. (2012). Applying static analysis in multi-threaded, large-scale java programs. Business Insider, 32(2), 23-25.

Kichel, Y.U. (2008). Ranking software inspection output using static profiling. Computer Applications Analysis, 34 (3), 34-45.

Moffat, P.W. (2010). Use of data flow analysis in static profiling. Software Business Publication, 34(2), 23-34.

Mosley, T., Beuby, W., & Walter, U. (2008). Correlation exploitation - Statistical Analysis. Analysis Symposium Workbook, 12(1), 234-245.

Myers, E.R. (2009). IEEE Standard for Software Analysis Reviews. Software Engineering Vocabulary, 23(1), 34-36.

Roldenson, P.O., & Waltz, E. (2003). Ranking software inspections and prioritizing analysis. Standard Software Conference, 23(4), 23-36.

Stan, Y.J., & Fowler, T. (2011). Dynamically discovering program invariants in supporting program evaluation. The Business Journal, 34(2), 123-145.

Simmon, T. (2010). A meta-analysis for effectively prioritizing errors in programming. Computer Science Journal, 23(3), 45-67.

Walter, Y. (2010). Writing dependable computer engineering research. Computer Engineering Journal, 34(4), 23-35.