Poster Abstract

Submitted by: Dalal Alrmuny
alrmuny@cs.colostate.edu
Advisor: Prof. James Bieman
bieman@cs.colostate.edu
Computer Science Department
Colorado State University

Testability Analysis of Object-Oriented Software

According to the 1990 IEEE standard, testability is “the degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met”. Testability attributes include the difficulty of reaching faults, the difficulty of generating test cases, and the likelihood that an existing fault will cause a noticeable failure.

Testability analysis is a method used for the quantitative or qualitative evaluation of the characteristics of software artifacts (i.e., control flows, data flows, and class dependencies) in order to assess its testability. The evaluation is based on inspecting the artifacts to gather information about certain properties. In object-oriented software, the existence of some types of intra- and inter-component interactions affects testability. The results of testability analysis are used to identify the properties of the artifacts that are critical to testability and what changes can be applied to improve testability.

In this poster, we show the main existing testability analysis approaches. Based on the artifact used in the analysis (i.e. source code or UML models), the analysis is classified as code-based or model-based. The poster compares the testability analysis approaches based on three criteria.

1. The testability attribute(s) addressed by the approach, and the metrics used to assess testability.
2. The way authors evaluate their analysis approach and the evidence they provide to show that the approach achieves its intended goal.
3. To what extent the analysis approach is practical, in terms of the analysis context (i.e., the underlying assumptions about the analysis subject), the required resources, and any extra overhead added to the compilation, program execution or testing of the software due to the analysis.

Finally, open problems are identified and listed. For example, many researchers left out the analysis of some complex - *though essential*- features of object-oriented software like polymorphism and inheritance. Furthermore, when these features were addressed, only the syntactic properties were considered without considering the semantics. Another open problem is the lack of a comparison between the different analysis approaches, which can serve as a guideline for the selection among them. We point the recent issues in testability analysis and the promising research directions.