Improving Testability for OO Programs through Design by Contract

Poster Abstract

Fatmah Y Assiri (Advisor: James M Bieman)

Computer Science Department

Colorado State University

Fort Collins, Colorado 80523

fatmahya@cs.colostate.edu

Design by contract (DbC) is a systematic approach to help provide reliable software that satisfies its specifications. Contracts indicate the expectations of the client and requirements of the supplier, and are written as assertions using Boolean expressions. Assertions can be associated with each method as pre-conditions and post-conditions. Class invariants specify the expected state before and after the executions of any class method. Contracts can be applied to source code or to UML (Unified Model Language) models.

The correctness of components in a software system helps to guarantee the correctness of the software as a whole. Applying DbC for procedural languages is fairly well-understood. Applying DbC to object-oriented (OO) programs introduces problems because of the presence of inheritance and other constructs. *Behavioral subtyping* contracts are based on the specifications of both the superclass and the subclass. They ensure that the specification of a subclass object satisfy the specification of the superclass objects. These contracts also must ensure that the calling methods provide the expected results.

In addition to problems when applying DbC to OO programs, there are challenges when applying DbC to concurrent programs. For example, *inferences* and *thread-safety specifications* are difficult, especially when inheritance relations are present. The *safepoint* approach is applicable for programs with concurrency. A *safepoint* is a safe place inside a method body to check pre-conditions and post-conditions because at this point the objects that are used in the contracts are protected with locks. The safepoint mechanism allows the safe and effective use of sequential contracts in programs with concurrency.

In my research I am studying measurable improvements in testability of OO software due to the use of DbC. DbC can improve testability by locating the area that holds faults. The IEEE defines testability as “a system design characteristic that allows its operational status to be determined and the isolation of faults to be performed efficiently”. Diagnosbaility , observability, and testing costs and effort are testability factors that I am using to assess testability.

The main approaches that I investigate in this research include the following:

- Baudry and Jezequel’s quantitative method to estimate improvements in vigilance and diagnosability when placing assertions.
- Briand and Labiche’s empirical study of testability improvements through the use of contracts. They used observability and disgnosability factors to assess testability.
- Cheon and Leavens’s Java-Modeling- Language (JML) to translate the software specifications into test oracles.
- Voas applied sensitivity analysis to determine low testable locations in the code and insert assertions in these parts to improve faults observability.

I identify some open problems including the need to study testability with respect to other testability factors. In addition to that, more work has to be done to evaluate the effectiveness of JML to write test cases and the use of *safepoints* in concurrency environments.

References

1. Briand, L., Labiche, Y., Sun, H.: Investigating the Use of Analysis Contracts to Improve the Testability of Object Oriented Code, Software: Practice and Experience (SPE) , 33 (7), pp. 637-672, 2003.
2. Le Traon, Y.; Ouabdessalam, F.; Robach, C. ; Baudry, B.; , From Diagnosis to Diagnosability: Axiomatization, Measurement and pplication, J. Systems and Software, vol. 65, no. 1, pp. 31-50, 2003.
3. Voas, J., Software testability measurement for intelligent assertion placement. Software Quality Control, 1997. 6(4): p. 327-336.
4. Yoonsik Cheon and Gary T. Leavens. A simple and practical approach to unit testing: The JML and JUnit way. In Boris Magnusson, editor, ECOOP 2002 — Object-Oriented Programming, 16th European Conference, M´aalaga, Spain, Proceedings, volume 2374 of Lecture Notes in Computer Science, pages 231–255, Berlin, June 2002. Springer-Verlag.
5. Le Traon, Y.; Baudry, B.; Jezequel, J.-M.; , Design by Contract to Improve Software Vigilance , Software Engineering, IEEE Transactions on , vol.32, no.8, pp.571-586, Aug. 2006 doi: 10.1109/TSE.2006.79