Using Metaprogramming to Implement a Test Framework

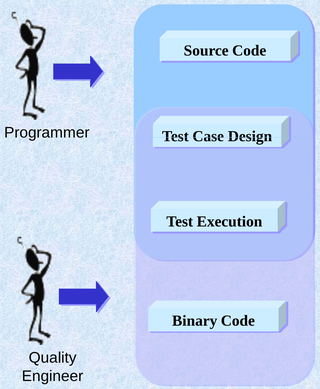
Hyun Cho
robusta@uab.edu

Advisor: Dr. Jeff Gray

Software Composition and Modeling Laboratory
SoftCom
Department of Computer and Information Sciences
University of Alabama at Birmingham

Metaprogramming and reflection are used to implement a common test framework that can help to reduce the burden for test preparation in unit and system testing. The framework focuses on generating instrumented Java bytecode based on several sources of input (e.g., source code or bytecode, and information provided by each test case).

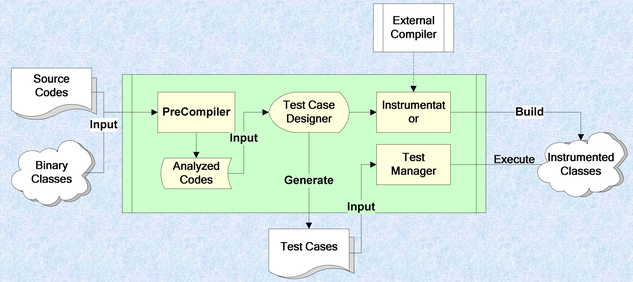
Challenges in Testing

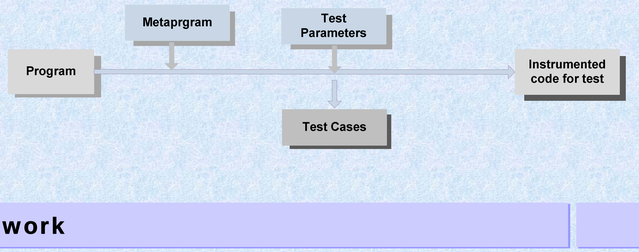


- Instrumentation of source code may need to be performed manually, which takes much time and is error-prone.
  → *Automate instrumentation using metaprogramming.*
- Different tools and methods are often used for unit and system testing. Thus, an organization must duplicate their investment of time and effort to maintain different types of testing tools and training their engineers.
  → *Provide unified Test Framework.*
- Test cases may be designed poorly (e.g., not complete or sufficient to test all behaviors). This negatively impacts the testing process and the quality of the tested software.
  → *Employ functional test on unit test and system test.*
- This project constructs a test framework based on metaprogramming to automate code instrumentation and to provide a unified test framework for unit/system testing.

Architecture Overview of Test Framework

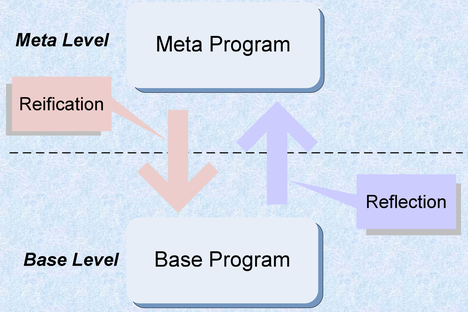
- Parser
  - ✓ Input processor and its implementation relies on metaprogramming.
  - ✓ Used to extract information such as classes, methods and attributes from the input source.
  - ✓ The type of Parser is automatically determined by considering the file type (e.g., .java for source code and .class for bytecode).
  - ✓ Selectively extract information so that it helps engineers to understand the system under test by limiting or expanding information per their needs.
- Test Case Designer
  - ✓ Represents a test case and the number of test cases can be governed after analyzing a test report.
  - ✓ Design a test case by entering values for arguments and the expected results of a test execution of a specific method.
  - ✓ Generate test case specification as a form of XML.





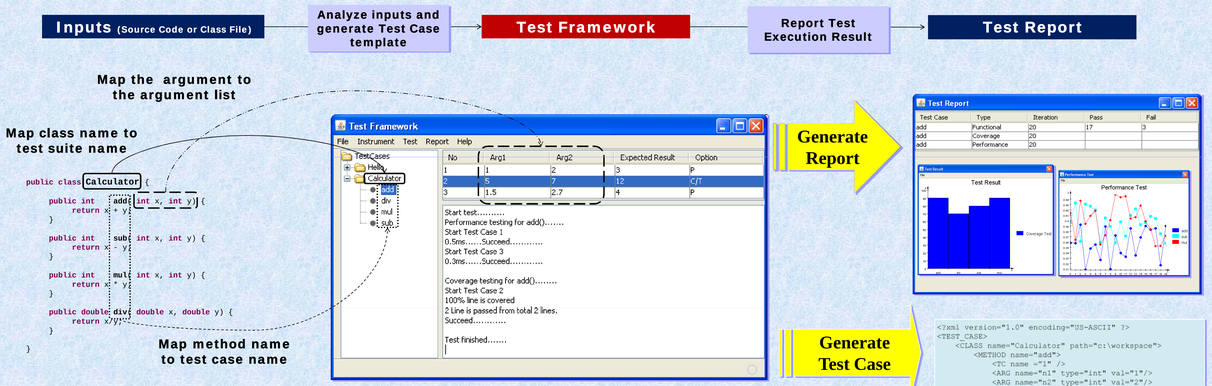
- Transformer
  - ✓ Generate instrumented binary codes, which produce run-tine information by loading each class file, retrieving its bytecode, and attaching metaobjects.
  - ✓ Responsible for checking the potential conflicts among the selected test options, e.g., coverage and trace vs. performance.
  - ✓ Two types of metaprograms are used to transform; Compile-time and Load-time
    - A compile-time metaprogram, like OpenJava[1], transforms source code through metaobjects.
    - A load-time metaprogram, such as Javassist[2] and JMangler[3], directly manipulates Java bytecode to generate instrumented test codes.
- Test Manager
  - ✓ Processes the test cases and controls the whole test execution by referring to the test case specification.
- External Compiler
  - ✓ Used when source codes are provided as input.

Metaprogramming



- **Reification** is the ability of a system to provide a concrete representation of its internal state.
- **Reflection**[4] represents a program's own execution state according to the reified meta objects in the meta space.
- **Metaprograms** are programs that modify other programs (often itself). There are two forms:
  - *Introspection*: query the structure and state of a program.
  - *Intercession*: ability to modify the structure or behavior of a program.

Test Framework



- Preliminary test case information is generated by parsing input codes.
- JUnit[5] is adopted to manage Test Suites and Test Cases for easy management and execution of test cases.
- XML is used to store test suites and test cases.
- Test result is also maintained using XML and provided in both graphical and table form.

Conclusion and Future work

- Metaprogramming can assist in the construction of a test framework (e.g., unit test and system test).
- The Parser and Transformer play key roles in this framework as a potential solution to the challenges addressed.
- Both compile-time and load-time metaprograms provide additional functionality transparently.
- The Transformer generates instrumented test code by attaching metaobjects so that instrumentation takes less time and helps to produce quality instrumented code.
- Future work
  - ✓ Effectively used on small applications but additional evaluation is needed by applying the test framework to a large open source application, such as JBoss or other system software written in Java.
  - ✓ Need to explore ways to reduce the amount of manual effort involved in specifying parameters in the Test Case Designer.

References

[1] Tatsubori, M., Chiba, S., Killijian, M., and Itano, K., "OpenJava: A Class-Based Macro System for Java," Reflection and Software Engineering, Denver, CO, November 1999, pp. 117-133.
[2] Chiba, S., "Load-time Structural Reflection in Java," European Conference on Object-Oriented Programming, Cannes, France, June 2000, pp. 313-336.
[3] Kniesel, G., Costanza, P., and Austermann, M., "JMangler: A Framework for Load-Time Transformation of Java Class Files," International Workshop on Source Code [Analysis and Manipulation, Florence, Italy, November 2001, pp. 100-110.
[4] Forman, I. and Forman, N., Java Reflection in Action, Manning Publication, 2004.
[5] http://www.junit.org/