Software Engineering

Prof. Dr. Bertrand Meyer

March 2007 – June 2007

Unit Testing

Gobo Eiffel Test and Clover
Agenda for Today

1. Testing
2. Main Concepts
3. Unit Testing – Gobo Eiffel Test
4. Test Evaluation – Clover
5. Reference
Software Testing

Goal: find many errors before shipping software
- Higher cost to fix errors after deployment
- Higher acceptance and confidence of users

Scientific approach
- Proof correctness and completeness of code

Pragmatic approach
- Try out software in typical usage scenarios

Fact
- Testing does not guarantee the absence of errors
Testing Scope

Testing in the small
- Exercising the smallest executable units of the system

Testing in the large
- Putting the entire system to the test
  
  Unit Testing
  Individual classes

  Component Testing
  Group of related classes

Integration Testing
Interaction between components
Unit Testing

Exercising the smallest individually executable units
Objectives: find faults in the units, assure correct functional behavior of units
Usually performed by programmers

The Typical Test Cycle
- Develop a suite of test cases
- Create test fixtures to support each test case
- Clean-up fixtures, if necessary
- Run the test and capture test results
- Report and analyze the test results
Testing Problem

Tests

Should write

Do

Programmers

few

I am so busy...

It is difficult...

Why?

Programmers need a tool to:

“Write a few lines of code, then a test that should run; or even better, write a test that won't run, then write the code that will make it run.”

Write a few lines of code, then a test that should run; or even better, write a test that won't run, then write the code that will make it run.”
Gobo Eiffel Test

A simple framework to write and run repeatable tests

Gobo Eiffel Test features include:

- Assertions for testing expected results
- Test fixtures for sharing common test data
- Test suites for easily organizing and running tests
- Textual test runners
Terminology

Test Case
- defines a method to run a set of tests

Test Suite
- a collection of related test cases

Test Fixture
- a common set of test data and collaborating objects shared by many tests
- Generally are implemented as instance variables in the test class

Test Runner
- runs tests and reports results

Errors and failures
- An error is some unanticipated failure (e.g., an exception thrown inside the tested code)
- A failure is anticipated, and is produced by a call of an assertXXX feature
Unit Testing Steps

Create a test class
- Declared as a subclass of TS_TEST_CASE

Create Test Case
- Name the test method as testXXX
- Asserts the expected results on the object under test

Use Test Fixture when necessary

Check for expected exceptions

Run the tests in the console
Gobo Eiffel Test Assertions

Within a test

- Call the method being tested and get the actual result
- Assert what the correct result should be with one of the provided assert methods
- These steps can be repeated as many times as necessary

An assert feature

- Is a **TS_ASSERTION_ROUTINES** feature that performs a test
Gobo Eiffel Test Assertions (Cont’d)

- `assert (a_tag: STRING; a_condition: BOOLEAN)`
  - Assert `a_condition`.
- `assert_equal (a_tag: STRING; expected, actual: ANY)`
  - Assert that equal (expected, actual).
- `assert_not_equal (a_tag: STRING; expected, actual: ANY)`
  - Assert that not equal (expected, actual).
- `assert_same (a_tag: STRING; expected, actual: ANY)`
  - Assert that expected = actual.
- `assert_arrays_same (a_tag: STRING; expected, actual: ARRAY [ANY])`
  - Assert that expected and actual have the same items
  - in the same order (use ‘=’ for item comparison).
- `assert_arrays_equal (a_tag: STRING; expected, actual: ARRAY [ANY])`
  - Assert that expected and actual have the same items
  - in the same order (use equal for item comparison).
- `assert_iarrays_same (a_tag: STRING; expected, actual: ARRAY [INTEGER])`
  - Assert that expected and actual have the same items
  - in the same order (use ‘=’ for item comparison).
Use of Fixtures

Some test cases act on similar sets of objects

- Create a fixture instead of declaring them in all methods
- Write as many Test Cases as you like
- Add as many test methods as you like

Use in detail

- Add fields for each part of the fixture
- Define `set_up` to initialize the fields
- Define `tear_down` to release any permanent resources
Configuration File

```
test
  my_test --name of the test suite

default
  class ("TEST_[A-Z0-9_]\*") --names of the test case classes
  feature ("test_[a-z0-9_]\*") --names of the test routine
  prefix ("X") --prefix to be used for the generated class names
  testgen ("TESTGEN") --specify the name of that directory where
    new classes should be generated
  compile ("compile se.ace") -- command-line instruction to be used
    to compile the test suite program
  execute ("my_test") -- command-line instruction to be used
    to run the test suite program

cluster
  test_dir:  "$GOBO/test/my_test"
  other_tests: "$GOBO/test/my_test/other"
    --the clusters containing the test case classes

end
```
Test Evaluation: Code Coverage

How good is a test?
Do we have enough test cases?
Testing is inherently incomplete

- **Coverage metrics**: quantitative evaluation of test suite
- **A test evaluation tool** helps in assessing whether the test cases achieve good coverage or not

Tools

- Clover, Quilt, Emma, Coverlipse, JDepend, Cobertura, Java Test Coverage, ...
Clover (Cover Lover)

Reports Test Coverage by
- Statement Coverage
- Branch Coverage
- Method Coverage

Uses Source Code Instrumentation
- Duplicates and modifies source code
- Requires separate compilation

Reports findings in multiple formats
- From project level down to individual lines of source code
- In XML, HTML, PDF formats

Integrated in NetBeans, Eclipse, and other IDEs

License: commercial (30-day evaluation available)
Using Clover Interactively

Typical cycle:
- write code and tests
- run tests
- inspect test results and code coverage
- repeat until all tests pass and code coverage of the tests meets a certain level
Clover coverage filters

One can choose not to instrument certain types of blocks in the code (e.g. assertions and exception catching), in order to focus on the coverage of interest.
Clover – Code Coverage

Clearly see how much is being tested

```
public boolean equals(Object anObject) {
    if (isZero())
        return ((IMoney)anObject).isZero();
    if (anObject instanceof IMoney)
        return ((IMoney)anObject).isZero();
    if (anObject instanceof MoneyBag)
        MoneyBag aMoneyBag = (MoneyBag)anObject;
    if (aMoneyBag.fMonies.size() != fMonies.size())
        return false;
    for (Enumeration e = fMonies.elements(); e.hasMoreElements();)
        Money m = (Money) e.nextElement();
```

NC Lines of Code: 113
Methods: 17
Files: -
Packages: -
Summary

“Any program feature without an automated test simply doesn’t exist”
Testable code improves confidence and design
Programmers can sleep better
“Keep the bar green to keep the code clean!”
Reference

Gobo Eiffel Test

JUnit
http://www.junit.org

Extreme programming
http://www.xprogramming.com

Clover
http://www.cenqua.com/clover/