Automated Tests Patterns

Paulo Cheque

Summer 2009

License:
Creative Commons: Attribution-Share Alike 3.0 Unported
http://creativecommons.org/licenses/by-sa/3.0/

Introduction

- Automated Tests have source code
- Require good development
- Susceptible to the same problems of the SUT source code
  - Requires maintenance
  - Requires a good design (simple design)
  - May have errors

Motivation

- Patterns => Common solution to recurrent situations

- Facilitate and optimize test writing

- Refactoring test code

Legend

```
// Comment
@Test
public void method() {
    String a = “some string”;
}
```

Definitions

- Patterns:
  - Description of a solution for recurrent problems in software development
- Anti-Patterns:
  - Not recommended patterns, which lead to development problems
- Smells/Evidences:
  - Symptoms in source code that suggest something is wrong

Smell

- Code
- Behavior
- Project

Smells in Code

- Obscure Test
- Conditional Test Logic
- Hard-to-Test Code
- Test Code Duplication
- Test Logic Production

Smells in Behavior

- Assertion Roulette
- Erratic Test
- Fragile Test
- Frequent Debugging
- Manual intervention
- Slow Tests

Smell in Project

- Buggy Tests
- Developer not Writing Tests
- High Test Maintenance Cost
- Production Bugs

Patterns

- Organization
- Results verification
- Value
- Set up
- Tear down

- Strategies
- Architecture

Organizational Patterns

- Methods:
  - Test Utility Method
  - Test Helper
  - Parametrized Test
  - Testcase Superclass
- Classes:
  - Testcase Class per Class/Feature/Fixture
- Suite:
  - Named Test Suite

Assumptions

- Never add test code in SUT
  - Don't use test logic in SUT
- Do not include test code in libraries
  - Pack in a independent file
- Standardize class names to facilitate identification and utilization of scripts
  - SomeClasse**Test**

Methods

- Test Utility Method / Parametrized Test
  - Refactoring: Extract Method
- Test Helper
  - Extract Method + Extract Class
  - Useful methods for more than one class
  - **SomeNameTestHelper**
- Testcase Superclass
  - Extract Method + Extract Superclass

Testcase Class per ...

- Class (First option):
  - SomeClass => SomeClassTest
- Feature (may be a method):
  - Organizing test classes with a lot of tests
  - Number of methods of a test class grows faster than SUT class
- Fixture:
  - Organizing according to Set Up

Named Test Suite

- Test groups
- Examples:
  - Test suite of smoke tests
  - Test suite of stored procedure tests
- TestNG (Java) has good support to this:

```
@Test(groups = {“SmokeTests”, “Interface”})
public void verifyBrokenLinks() { … }
```

Results Verification

- State Verification
- Behavior Verification
- Custom Assertion
- Delta Assertion
- Guard Assertion
- Unfinished Test Assertion

Verification

- State Verification
  - assertEquals(expectedState, sut.getState())
- Behavior Verification
  - No state (logs), use Spy or Mock

- Be careful with Anti-Pattern: Verify the implementation, not the feature

```
expect(mock).doIt()).times(53)
```

Custom Assertion

```
public class IsNotANumber extends TypeSafeMatcher<Double> {
  @Override public boolean matchesSafely(Double number) {
    return number.isNaN();
  }
  public void describeTo(Description description) {
    description.appendText("not a number");
  }
  @Factory public static <T> Matcher<Double> notANumber() {
    return new IsNotANumber();
  }
}
assertThat(Math.sqrt(-1), is(notANumber()));
```

Assertion

- Delta Assertion
  - assertEquals(list.size()+1, list.size())
- Guard Assertion
  - Verification by caution
- Unfinished Test Assertion
  - @Ignore or skip
  - fail("Not yet implemented")

Value Patterns

- Literal Value
  - *Hard-coded* information: BigDecimal("13.42")
- Derived Value
  - Implement an algorithm that generates the expected value
  - Be careful to not create the same algorithm in tests

Value Patterns

- Generated Value
  - Generate distinct values to each test
  - Useful for integration tests
  - Be careful with the reproducibility of tests

```
id = IDGenerator.uniqueID();
```

- Dummy Object

```
expect(obj.toString()).andReturn(anything());
```

Setup Patterns

- Fresh Fixture Setup
- Shared Fixture Construction

Setup: Clean

- In-line setup
  - Each test has a particular set up
- Delegated setup
  - Distinct SetUp to each test
  - Extract Method
- Implicit setup
  - TestCase per Fixture
  - Framework patterns

Setup: Shared

- Prebuilt fixture
- Lazy setup
- Suite fixture setup
- Setup decorator
- Chained tests
  - Add => Select => Update => Delete

Teardown Patterns

- Teardown Strategy
  - Garbage-Collected Teardown
  - Automated Teardown
- Code Organization
  - In-line Teardown
  - Implicit Teardown: Default

Teardown: Strategies

- Garbage-Collected Teardown
  - Delete
  - frame.cleanUp()
  - System.gc()

- Automated Teardown
  - Register of what was inserted to be cleaned in teardown

Teardown: Organizational

- In-line Teardown
  - Each test has a particular teardown
  - After verifications
- Implicit Teardown
  - Framework patterns

Strategies

- Record
- Scripts
- Tests with frameworks

- Data driven
- Spy (Back Door)
- Layers

Testable Architecture

- Dependency Injection
- Find dependencies(lookup)
- Humble Object
- Test Hook (Anti-Pattern)

Dependency Injection

- Implementation
  - Constructor
  - Setter / public variables
  - Lookup

- Object-Oriented:
  - Tell, do not ask!

Humble Object

- Object has important rules to be tested, but the rules are interlaced with some complex code such as untestable frameworks tasks.

- Refactoring:
  - Separate responsibilities:
    - A method contains the rules
    - Another method will be an adapter (Humble Object) that will delegate rules to other object

Test Hook

- Modify the system to work differently in test execution

```
if(TESTING) { ... } else { ... }
```

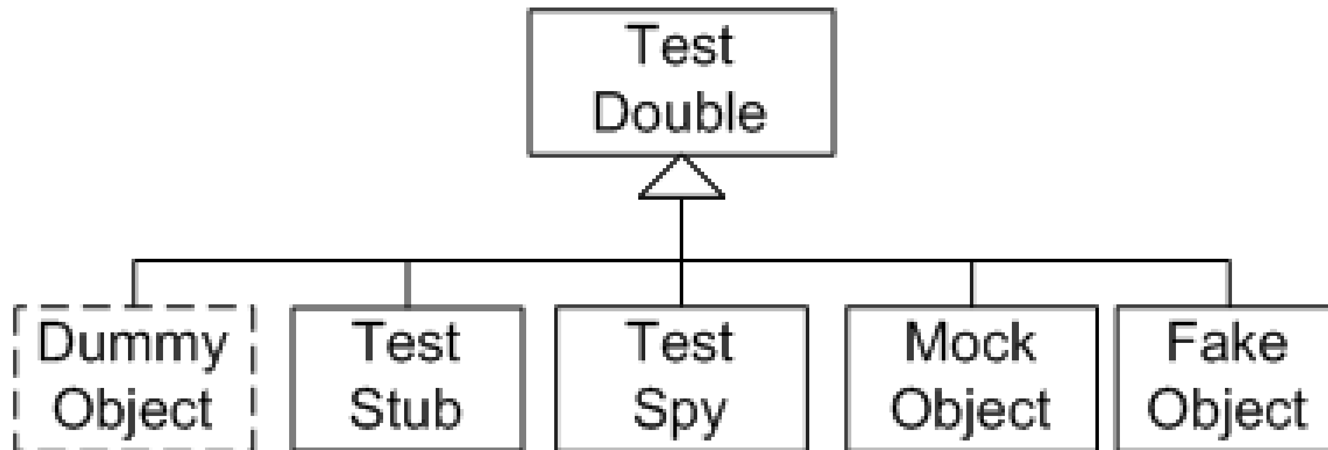
- Anti-Pattern: Add test code in SUT

Anti-Patterns

- Change SUT to pass the test
- To adequate the test to SUT, not SUT to test
- Test the implementation, not the feature

- Anti-Patterns in source code
- Smell => Anti-Pattern

Double



Contact

http://www.agilcoop.org.br

http://ccsl.ime.usp.br

http://qualipso.org

agilcoop@agilcoop.org.br

paulocheque@agilcoop.org.br

License:
Creative Commons: Attribution-Share Alike 3.0 Unported
http://creativecommons.org/licenses/by-sa/3.0/