Visualizing a test suite's composition and relevance

Bart Van Rompaey
Lab On Reengineering
University of Antwerp

Benevol5
Dec. 11, 2006
The benefits of unit testing in terms of defect prevention and software quality of unit testing are well known [Maximilien&Williams][Crispin]

Studies report a ratio of test to production code ranging between 2:3 and 1:1 [Moonen et al.][Sangwan&Laplante]

Program understanding can take up more than half of the resources spent to maintenance and evolution [Bennett&Rajlich]
• Motivation
• Objectives
• Modeling Software Tests
• Visualization Techniques
  - A System-wide test suite visualization
  - UUT/Test Case Specific
• Conclusion & Future Work
Objectives

Assist the developer in identifying and evaluating

- the physical location of test code
- structural characteristics of the test suite
- relevant test cases in an evolutionary context
- test maintainability

by visualizing a test suite in terms of common testing concepts
Concepts of Unit Testing

Automated

Independent

Persistent

Stable

Fast

Repeatable

Concise

Self-Checking

Isolated

Universiteit Antwerpen

(http://xunitpatterns.com/Organization.html)
public class CarTest extends TestCase {
    private Car aCar;

    public void setUp() {
        aCar = new Car();
    }

    public void testCar() {
        assertTrue(aCar.getVelocity() == 0);
    }

    public void testAccelerate() {
        aCar.accelerate();
        assertTrue(aCar.getVelocity() == 1);
    }

    public void testBrake() {
        aCar.accelerate();
        aCar.brake();
        assertTrue(aCar.getVelocity() == 0);
    }
}

public class Car {
    private int fVelocity;

    public Car() {
        fVelocity = 0;
    }

    public void accelerate() {
        fVelocity++;
    }

    public void brake() {
        fVelocity--;
    }

    public int getVelocity() {
        return fVelocity;
    }
}
Modeling Test Code

- Test model as a refinement of a standard OO metamodel
System-wide test suite visualization

Ant-1.6.5 / JUnit

edges:
- green = containment
- red = tests relation
- purple = test dependency
Test Suite Patterns

Untested code

Indirect Tests

Universiteit Antwerpen

Test Dependencies

Isolated tests

Dependent tests

org.apache.tools.ant.BuildFileTest

org.apache.tools.ant.Project
System-wide visualization: lessons learned about Ant

- Test cases are located in the same package as the unit under test
- Certain production classes are used by most test cases (Project, Path, File).
- **BuildFileTest** is a test helper commonly used
- Only a limited amount of components are tested in isolation (e.g. the cvs module)
Test Case Visualization

- Idea: visualize Test Cases and Units under Test in terms of Fixture, Test Commands, Stimuli, ...
Relevant Test Cases

not tested

methods
under test

1 relevant
test case

Universiteit Antwerpen
Assessing structure/quality of test cases

make part of fixture?

not tested

interesting scenarios?

Universiteit Antwerpen
Limitations

• Due to static analysis:
  - Polymorphic calls not correctly resolved
  - Object fixtures (e.g. ArgoUml)

• Due to tool prototype
  - bound to JUnit
  - slightly adapt test entity detection heuristics (?)
Future Work

Add measurements to graph entities
- polymetric nodes for size, complexity, test smell metrics
- edge thickness/color/direction could represent e.g.
  - nr. of calls
  - call trace information
- ... (what would you like to add?)
Questions

bart.vanrompaey2@ua.ac.be