Issues in Object-Oriented Testing

Testing Extravaganza Weekend

James Gawn

02.12.2006
What am I going to talk about?

- A little background of object orientated programming
- Unit testing issues
- Implications various object orientated properties:
  - Composition and Encapsulation
  - Inheritance
  - Polymorphism
- Levels of object orientated testing
- A quick round up
The Object Orientated Paradigm

- Started way back in 1960’s with PDP-1 System from MIT
- Smalltalk in the 1980’s influenced the introduction of the idea of inheritance
- Became more widely used in 1990’s with advent of C++
- Promised to make code reuse easier
- Unfortunately does introduce a new set of issues for testing
- Treats programming as a series of co-operating objects, opposed to collections of functions
Overview of Object Orientated Unit Testing
What is a unit in an object orientated system?

- Traditional systems define a unit as the smallest component that can be compiled and executed.
- Units are normally a component which in theory is only ever assigned to one programmer.

Two options for selecting units in object orientated systems:

- Treat each class as a unit.
- Treat each method within a class as a unit.
Advantages for Object Orientated Unit Testing

- Once a class is testing thoroughly it can be reused without being unit tested again
- UML class state charts can help with selection of test cases for classes
- Classes easily mirror units in traditional software testing
Disadvantages for Object Orientated Unit Testing

- Classes obvious unit choice, but they can be large in some applications
- Problems dealing with polymorphism and inheritance
Implications of Composition and Encapsulation
Composition Issues

- Objective of OO is to facilitate easy code reuse in the form of classes
- To allow this each class has to be rigorously unit tested
- Due to classes potentially used in unforeseeable ways when composed in new systems
  - Example: A XML parser for a web browser
- Classes must be created in a way promoting loose coupling and strong cohesion
Encapsulation Issues

- Encapsulation requires that classes are only aware of their own properties, and are able to operate independently.
- If unit testing is performed well, the integration testing becomes more important.
- If you do not have access to source code then structural testing can be impossible.
- If you violate encapsulation for testing purposes, then the validity of test could be questionable.
Implications of Inheritance and Polymorphism
The Issues

- Inheritance is an important part of the object-oriented paradigm
- Unit testing a class with a super class can be impossible to do without the super classes' methods/variables

```java
public class Fruit {
    private int weight;

    public Fruit() {
        // Grow a piece of fruit
    }

    public void prepare() {
        // Wash Fruit
    }

    public void eat() {
        // Consume the piece of fruit
    }
}

public class Orange extends Fruit {
    private int numberOfSegments;

    public void prepare() {
        // Peel Fruit
        peel();
    }

    private void peel() {
        // Peel orange
    }
}
```
One Solution - Flattening

- Merge the super class, and the class under test so all methods/variables are available
- Solves initial unit test problems
- Problems:
  - The class won’t be flattened in the final product so potential issues may still arise
  - Complicated when dealing with multiple inheritance
One Solution - Flattening

```java
public class Orange{
    private int noOfSegments;
    private int weight;

    public Orange() {
        //Grows a piece of fruit
    }

    public void prepare() {
        //Peel Fruit
        peel();
    }

    private void peel() {  
        //Peel orange
    }

    public void eat() {  
        //Consume the piece of fruit
    }
}
```
Polymorphism Issues

- Repeatedly testing same methods
- Time can then be wasted if not addressed
- Potentially can be avoided, and actually save time

```java
public class Fruit {
    private int weight;
    public Fruit() {
        // Grows a piece of fruit
    }
    public void prepare() {
        // Wash Fruit
    }
    public void eat() {
        // Consume the piece of fruit
    }
}

public class Orange extends Fruit {
    private int numOfSegments;
    public void prepare() {
        // Peel Fruit
        peel();
    }
    public void peel() {
        // Peel orange
    }
}

public class Strawberry extends Fruit {
    private int redness;
    // Returns how red the strawberry is.
    public int returnRedness() {
        return redness;
    }
}
```
Polymorphism Issues - Example Diagram
Polymorphism Issues - Example Code

```java
public class Strawberry extends Fruit {
    private int weight;
    private int redness;

    public Strawberry() {
        // Grows a strawberry
    }

    public void prepare() {
        // Wash Fruit
    }

    // Returns how red the strawberry is
    public int returnRedness() {
        return redness;
    }

    public void eat() {
        // Consume the piece of fruit
    }
}
```

```java
public class Orange{
    private int noOfSegments;
    private int weight;

    public Orange() {
        // Grows a piece of fruit
    }

    public void prepare() {
        // Peel Fruit
        peel();
    }

    private void peel() {
        // Peel orange
    }

    public void eat() {
        // Consume the piece of fruit
    }
}
```
Levels of Object Orientated Test

- There are generally 3 or 4 levels of testing for object orientated systems depending on your approach, consisting of:
  1. Method Testing (Unit Testing)
  2. Class Testing (Unit Testing/Intraclass Testing)
  3. Interclass Testing (Integration Testing)
  4. System Testing
Summary
A Quick Roundup

- Object orientated testing is a pain
  - Encapsulation
  - Inheritance
  - Polymorphism
- Necessary evil as it is more widely adopted
- Light at the end of the tunnel is unit/class reuse
Issues in Object-Oriented Testing

Testing Extravaganza Weekend

James Gawn

02.12.2006