White-box testing

Software Reliability and Testing - Barbara Russo

SERG - Laboratory of Empirical Software Engineering

unibz
Freie Universität Bozen
Libera Università di Bolzano
Università Liedia de Bulsan

White-box testing

- White-box testing is a verification technique that software engineers can use to examine if their **software** works as expected

With WBT

- Exercise independent paths within a module or unit
- Exercise logical decisions on both their true and false side
- Execute loops at their boundaries and within their operational bounds and
- Exercise internal data structures to ensure their validity

Types of WBT

- Unit testing
- Regression testing
- Integration testing

Unit testing

- Each time you write a code module, you should write test cases for it
  - A possible exception: accessor methods (i.e., getters and setters)
    - generally, accessor methods will be written error-free
  - You should focus your testing effort on code that could easily be broken

Unit testing

- It focuses on faults within modules

Integration testing

- It verifies interface specifications and model breakdown
- It verifies the software architecture
- Scaffolding (see later) is extensively required
- **Faults** are related to interactions and compatibility

Integration Faults

- Inconsistent interpretation of parameters or values
  - Example: Mixed units (meters/yards) in Martian Lander
- Violations of value domains, capacity, or size limits
  - Example: Buffer overflow

2007 Mauro Pezzè & Michal Young

Integration Faults

- Side effects on parameters or resources
  - Example: Conflict on (unspecified) temporary file
- Omitted or misunderstood functionality
  - Example: Inconsistent interpretation of web hits
- Nonfunctional properties
  - Example: Unanticipated performance issues

Integration Faults

- Dynamic mismatches
  - Example: Incompatible polymorphic method calls

Regression testing

- Premise
- Adding new/changing module impacts a system:
  - New data flow paths established
  - New I/O may occur
  - New control logic invoked

Regression testing

- It is re-execution of subset of tests that have already been run
  - Ensures changes have not propagated unintended side effects

Regression testing

- Approaches:
  - Manual testing
  - Capture/Playback tools:
    - 1. Capture test cases and results
    - 2. Playback and
    - 3. Compare

Regression testing

- Test suite contains:
  - Representative sample of tests that exercises all software functions
  - Focus on functions likely affected by change
- Focus on components that have been changed

Scaffolding

Why Scaffolding?

- Test case design includes input and expected output behaviour
- It may be simply a matter to fill a template (Acceptance test), but …

Why Scaffolding?

- **Testing can be complex:** a test specification is connected to several test cases:
  - e.g., a sorted sequence, length greater than 2, with items in ascending order with no duplicates
- We need a structure to support test execution

Why Scaffolding?

- **To test in small:** independent drivers just test some functionalities of a large user interface
- **To provide hooks** for scripting
- **To drive coding** (TDD)

Elements

- Test driver
- Test hub
- Test harness
- Oracle

Test driver

- To drive program under test through test cases
- It is a software module used to **invoke a module under test** and,
- It often provides test inputs, control and monitor execution, and report test results
- Drivers can become automated test cases

Example

movePlayer(Player1, 2);

- It calls movePlayer and passes Player1 and moves it two spaces
- The drivers would likely to be called from main method
- A white-box test case would execute this driver line of code and check Player.getPosition() to make sure the player is now on the expected cell on the board

Test stub

- A stub is a program statement **substituting for the body of a software module** that is or will be defined elsewhere or
- A dummy component or object used to **simulate the behaviour of a real component** until that component has been developed

Test stub

- Developing stubs allows programmers to call a method in the code being developed, even if the method does not yet have the desired behaviour
- Stubs can be “filled in” to form the actual method

Example

- If the movePlayer() has not been written yet

```
public void movePlayer(Player aPlayer, int diceValue) {
    player.setPosition(1);
}
or

public Cell movePlayer(Player aPlayer, int diceValue) {
   return new Cell();
 }
 or

 public Cell movePlayer(Player aPlayer, int diceValue) {
     Player player=aPlayer;
     player.setPosition(diceValue);
     return player.getPosition(player);
    }
```

Test harness

- Environment in which to **execute the tests**
- Substitutes for other parts of the deployed environment
  - Ex: Software simulation of a hardware device

Oracle

- A test oracle is a piece of software that **provides a pass/fail service** to the program execution (e.g., *assert*)
- In principle, an oracle **classifies every execution and detects every failure**, but it can even **give false alarms**
  - False alarms increase cost of maintenance and reduce resources to dedicate to real failures
  - Thus, *there is no perfect oracle*

Using stubs and drivers with TDD

Software Reliability and Testing - Barbara Russo
SERG - Laboratory of Empirical Software Engineering

What are drivers, stubs, and oracles?

```
public void testCenterLine() {
      Formatter f = new Formatter();
      f.setLineWidth(10);
      assertEquals("   word    ", f.center("word"));
}
public void testOddCenterLine(){
      Formatter f = new Formatter();
      f.setLineWidth(10);
      assertEquals( "  hello    ", f.center("hello"));
}
```

```
public String center(String line) {
    return “a”;
}
```

What are drivers, stubs, and oracles?

```
public void testCenterLine() {
      Formatter f = new Formatter();
      f.setLineWidth(10);
      assertEquals("   word   ", f.center("word"));
}
public void testOddCenterLine(){
      Formatter f = new Formatter();
      f.setLineWidth(10);
      assertEquals( "  hello    ", f.center("hello"));
}
```

```
public String center(String line) {
    return “a”;
}
```

More on Oracles

Software Reliability and Testing - Barbara Russo
SERG - Laboratory of Empirical Software Engineering

unibz
Freie Universität Bozen
Libera Università di Bolzano
Università Liedia de Bulsan

Comparison-based oracles

- **To judge oracle correctness**: compare the actual with predicted program behaviour
- **Predicted behaviour**: either pre-computed as part of test case specifications (when test specifications are simple) or derived in some way independent from the program under test

Comparison-based oracles

- Test harnesses often include support for comparison -based oracles: frameworks like JUnit provide methods like assertEquals

Comparison-based oracles

- Comparison-based oracles can be suitable for more complex programs when supported by other techniques like the “capture and replay” testing
  - **Capture - replay** testing saves the predicted behaviour from an earlier execution
  - It useful **when we cannot derive the expected behaviour** from specification
  - It can be **expensive** as it requires other implementations of the same feature under test

Example

- Save the behaviour (execution) of a trusted alternate version of the program under test
- For some reason, the current version is not adequate for the product under test
  - Think of OSS development: the open and the enterprise release
  - Replay the open version to test the enterprise one

Example

- Save system behaviour (use): used when behaviour cannot avoid the direct intervention of the user
- Use log of use: they can be reused until the programmer changes that functionality and cannot ensure the same behaviour anymore
  - Sometime useful to model the input in a more abstract way (e.g. instead of mouse click use object selection)

Other approaches for complex input-output

- Determine input by knowing the output:
  - It is a misconception that oracles always require a predicted output!

Example - partial oracles

- If a program is required to find a bus route from A to B, the oracle needs not to check that a there is a **valid** route between A and B
- If a program is required to find the optimal route between A and B, the oracle needs not to check that the valid route is also **optimal**
- These oracles are called partial
- Partial oracles are cheap

Self-check oracles

- Self-check: an oracle that **is not paired** with a specific test case
- Good design practice: not connected with program states
- Self-checks can use assertions and can focus on data structure invariants

Example

```
package org.eclipse.jdt.internal.ui.text;
import java.text.CharacterIterator;
import jorg.eclipse.jface.text.Assert;
 ...
public class SequenceCharacterIterator implements
CharacterIteratorf{
   //Check the index of iterations; self-check oracle
    private void invariant(){
      Assert.isTrue(fIndex >= fFirst);
      Assert.isTrue(fIndex <= fLast);
   }

it continues …
```

Example

```
…
public SequenceCharacterIterator(CharSequence
sequence, int first, int last) throws
IllegalArgumentException{
  if(sequence==null)
     throw new NullPointerException;
  if(first<0 || first>last)
     throw new IllegalArgumentException;
  if(last>sequence.length())
     throw new NullPointerException;
  fSequence = sequence;
  fFirst = first;
  fLast = last;
  fIndex = index;
  invariant();//Check the index of iterations
}
```

Example

```
...
 public char setIndex(int position){

   if(position>=getBeginIndex() && position <= getEndIndex())
      findex=position;
   else
       throw new IllegalArgumentException();
   invariant(); //Check the index of iterations
   return current();
 }


}
```