Software Engineering: Theory and Practice

Verification by Testing

Test Case Design

Tom Verhoeff

Eindhoven University of Technology
Department of Mathematics & Computer Science
Software Engineering & Technology

Feedback to T.Verhoeff@TUE.NL

© 2008–2012, T. Verhoeff @ TUE.NL
Do Not Confuse Testing and Debugging

**Testing** = The process of executing software with the intent of detecting the presence of defects. Works indirectly, through failures; often does not localize defects. Testing determines a measure for quality. Testing is only one of many verification activities.

**Debugging** = The act of fault diagnosis and correction. Debugging concerns rework. Debugging is time consuming and unpredictable.
The problem is the testing of the following program:

The program reads three integer values from a card.

The three values are interpreted as representing the lengths of the sides of a triangle.

The program prints a message that states whether the triangle is scalene, isosceles, or equilateral.

Write a set of test cases that you feel would adequately test this program.

Self-Assessment Test Scoring

1. **Valid scalene triangle included?**
   - **OK** \((3, 4, 5)\).
   - **NO** \((1, 2, 3)\) or \((2, 5, 10)\).

2. **Valid equilateral triangle included?**
   - **OK** \((3, 3, 3)\).
   - **NO** \((0, 0, 0)\).

3. **Valid isosceles triangle included?**
   - **OK** \((3, 3, 1)\).
   - **NO** \((2, 2, 4)\).
4. All three permutations of valid isosceles triangle?
   **OK (3, 3, 1) and (3, 1, 3) and (1, 3, 3).**

5. One side equal zero?
   **OK (0, 4, 5).**

6. One side negative?
   **OK (−3, 4, 5).**
7. **Degenerate triangle \((a + b = c)\)?**
   **OK** \((1, 2, 3)\).

8. **All three permutations of degenerate triangle?**
   **OK** \((1, 2, 3) \text{ and } (2, 3, 1) \text{ and } (3, 1, 2)\).

9. **Non-triangle with positive sides \((a + b < c)\)?**
   **OK** \((1, 2, 4)\).

10. **All three permutations of non-triangle?**
    **OK** \((1, 2, 4) \text{ and } (2, 4, 1) \text{ and } (4, 1, 2)\).

11. **All sides zero?**
    **OK** \((0, 0, 0)\).

© 2008–2012, T. Verhoeff @ TUE.NL

Software Engineering: Testing
12. **Non-integer values?**
   \( \text{OK (}'A', 'B', 'C'\text{)} \).

13. **Wrong number of values?**
   \( \text{OK (}3, 4\text{) or (}3, 4, 5, 6\text{).} \)

14. **Expected output for each case included?**
Some Testing Principles

- A necessary part of a test case is a definition of the expected output or result.

- Thoroughly inspect the result of each test.

- Avoid throw-away test cases unless the program is truly a throw-away program.

- Do not plan a testing effort under the tacit assumption that no faults will be found.

- Testing is an extremely creative and intellectually challenging task.
Levels of Testing in V-Model (from ESA SE Std)
What Qualities to Test

• **Utility**: To what extent is required functionality provided?

• **Reliability**: To what extent does the product fail?
  How frequently, how critical?

• **Robustness**: What happens in unexpected situations?

• **Efficiency**: How much is used of resources? Time, memory, disk, network, . . .

• **Usability**: How easy is the product to use?
Approaches to Test Case Design

Black-box, or test-to-specifications, or functional:

Checks the functionality of the software.
Consider specification/requirements only. Ignore code.

Glass-box, or test-to-code, or structural:

Checks the internal logic of the software.
Consider code only. Ignore specification/requirements.
Techniques for Constructing Test Cases

- Boundary analysis
- Equivalence classes
- Statement, branch, and path coverage
### Coverage: Example

```
if C then v := 1
; if D then w := 2
else w := 3
```

5 (!) statements, 2 + 2 branches, 2 * 2 paths

<table>
<thead>
<tr>
<th>Test Cases</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Statement</td>
</tr>
<tr>
<td>¬C, ¬D</td>
<td>60%</td>
</tr>
<tr>
<td>C, D</td>
<td>80%</td>
</tr>
<tr>
<td>C, D, ¬D</td>
<td>100%</td>
</tr>
<tr>
<td>C, D, ¬C, ¬D</td>
<td>100%</td>
</tr>
<tr>
<td>C, D, ¬C, ¬D</td>
<td>100%</td>
</tr>
</tbody>
</table>

© 2008–2012, T. Verhoeff @ TUE.NL
Coverage: Example

Python code:

1 \( p, q = 0, N \quad \# \text{given } A[0..N] \)
2
3 \textbf{while} \( p <> q \) :
4 \hspace{1em} \textbf{if} \ A[p] : p = q
5 \hspace{1em} \textbf{else} : p = p + 1
6
7 \textbf{if} \ p == N : \texttt{print} \ "Not found"
8 \textbf{else} : \texttt{print} \ "Found at", p
9
10 \texttt{@} \ (0 <= p < N \ /\ A[p] \ /\ \) 
11 \texttt{@} \ (\forall q: q<p: \text{not } A[q]) \) 
12 \texttt{@} \ /\ \ p = N

What test cases to include?

© 2008–2012, T. Verhoeff @ TUE.NL
Testing Advice

- Develop test cases before coding (Test-Driven Development).
- Test incrementally (not everything together at once).
- Test simple parts first.
- Use assertions (built-in tests; “fail early”): Test pre- and post-conditions, and ‘can’t-happen’ cases.
- Automate testing.
- Keep test software, data, and results (commit in repository).
- Re-test after making changes (regression testing).
Test case: control activation and input; observe response and output; decide on pass/fail.

JUnit Automated Testing Framework

JUnit: organizes code for test cases, runs them, reports results

See NetBeans IDE sample program Anagrams (via New Project).

Help > Javadoc References > JUnit API

Test case: method named test...

Facilities: fail, assertTrue, assertEquals, ...

Right-click Java file in NetBeans project: Tools > Create JUnit Tests

Can also test for required exceptions: no/wrong exception → failure

© 2008–2012, T. Verhoeff @ TUE.NL 14 Software Engineering: Testing
Code Coverage Analysis

In NetBeans, you can use **JaCoCo** to analyze code coverage:

- [http://plugins.netbeans.org/plugin/48570/tikione-jacocoverage](http://plugins.netbeans.org/plugin/48570/tikione-jacocoverage)

- **Documentation:** [http://www.eclemma.org/jacoco/](http://www.eclemma.org/jacoco/)

- Right-click project, **Test with JaCoCoverage**

- Opens browser page with analysis results
References


• [JUnit Testing Framework](#) (integrated into the [NetBEans IDE](#))