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`

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.

Self-Assessment Test

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.

Glenford J. Myers. *The Art of Software Testing*. Wiley, 1979.

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** $(2, 2, 4)$.

Self-Assessment Test Scoring

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 $(-3, 4, 5)$.

Self-Assessment Test Scoring

7. Degenerate triangle $(a + b = c)$?
   OK $(1, 2, 3)$.

8. All three permutations of degenerate triangle?
   OK $(1, 2, 3)$ **and** $(2, 3, 1)$ **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 $(1, 2, 4)$ **and** $(2, 4, 1)$ **and** $(4, 1, 2)$.

11. All sides zero?
    OK $(0, 0, 0)$.

Self-Assessment Test Scoring

12. Non-integer values?
    OK ('A','B','C').

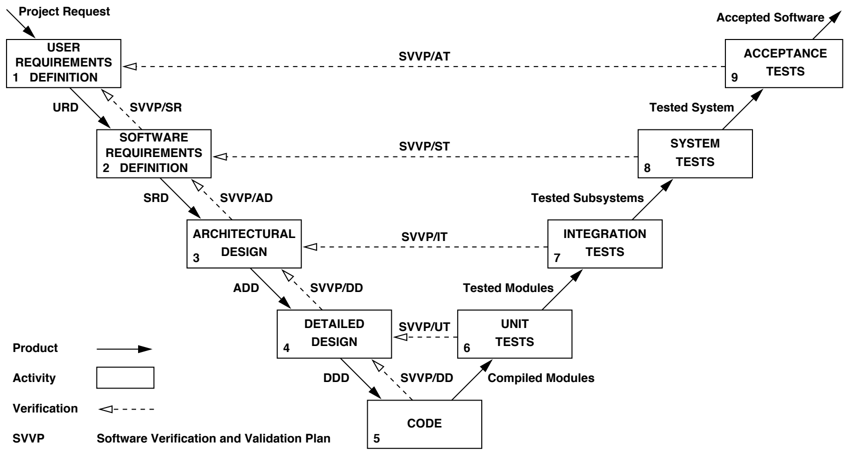
13. Wrong number of values?
    OK $(3, 4)$ **or** $(3, 4, 5, 6)$.

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)



Project Request

USER REQUIREMENTS 1 DEFINITION

URD

SVVP/SR

SOFTWARE REQUIREMENTS 2 DEFINITION

SRD

SVVP/AD

ARCHITECTURAL 3 DESIGN

ADD

SVVP/DD

DETAILED 4 DESIGN

DDD

SVVP/DD

CODE 5

Compiled Modules

UNIT 6 TESTS

SVVP/UT

Tested Modules

INTEGRATION 7 TESTS

SVVP/IT

Tested Subsystems

SYSTEM 8 TESTS

SVVP/ST

Tested System

ACCEPTANCE 9 TESTS

SVVP/AT

Accepted Software

Product

Activity

Verification

SVVP Software Verification and Validation Plan

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

Test Cases				Coverage		
1	2	3	4	Statement	Branch	Path
$\neg C, \neg D$				60%	50%	25%
C, D				80%	50%	25%
C, D	$C, \neg D$			100%	75%	50%
C, D	$\neg C, \neg D$			100%	100%	50%
C, D	$C, \neg D$	$\neg C, D$	$\neg C, \neg D$	100%	100%	100%

Coverage: Example

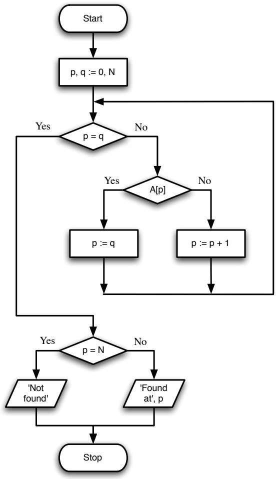
Python code:

```
p, q = 0, N   # given A[0..N)

while p <> q :
  if A[p] : p = q
  else : p = p + 1

if p == N : print "Not found"
else : print "Found at", p

#@ (0 <= p < N /\ A[p] /\
#@ (forall q: q<p: not A[q]))
#@ \/  p = N
```

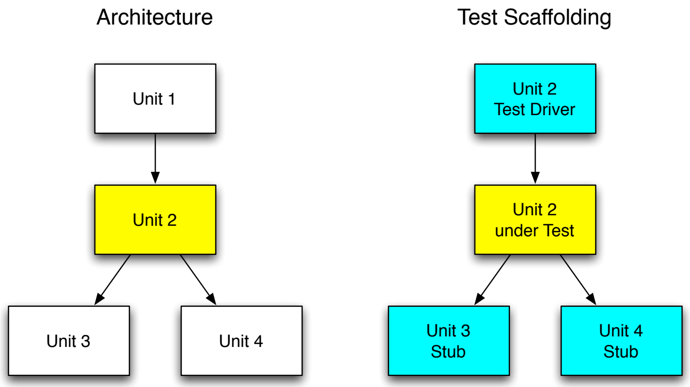


What test cases to include?

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).

Testing Terminology



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`, `assertEqual`, ...

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

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

References

- "What is Software Testing? And Why Is It So Hard?" by J. A. Whittaker in *IEEE Software*, **17**(1):70–79 (Jan./Feb. 2000).
- *Code Complete*, 2nd Ed. by Steve McConnell. Microsoft Press, 2004.
- JUnit Testing Framework (integrated into the NetBEans IDE)