Systematic Software Test (I)

Anne Haxthausen

Informatics and Mathematical Modelling

Technical University of Denmark

Overview

- Programs often contain unintended errors — how do you find them?
- Software test is an activity the goal of which is to reveal these.
- Two types of systematic test:
  - Structural test
  - Functional test

Parts of these foils are based on the lecture notes and foils on *Systematic Software Test* by Peter Sestoft.

Structural test versus functional test

Structural test is also known as internal test or white-box test.
Functional test is also known as external test or black-box test.

	Structural test	Functional test
Starting point	the source code	the problem
Found kinds of errors	logical errors wrong initialization of variables	unobserved cases unobserved requirements

Structural test and functional test

Test case = input data + expected output data

1. Design a test:
   - a table of input data and the corresponding, expected output data
   - a table of input data properties (relates program/problem to the test cases).
2. Execute the test cases:
   (a) run the program with the input data
   (b) compare actual output data with the expected output data

Structural test: Conditionals and loops

Design the test cases such that all parts of the program will be executed.

Statement	Cases to test
`if`	Condition false and true
`switch`	Every branch must be executed
`for`	Zero, one, and more than one executions
`while`	Zero, one, and more than one executions
`do-while`	One and more than one executions

Structural test: Composite logical expressions

Test all possible combinations of truth values for terms.

`(x != 0)`	`&&`	`(1000/x > y)`
false		
true		false
true		true

`(x == 0)`	`||`	`(1000/x > y)`
true		
false		false
false		true

Example: requirements

Write a program that takes some integers as input and prints the two smallest of these, or the smallest in case there is only one.

Example: implementation (MinTwo.java)

```
public static void main (String[] args) {
  int mi1 = 0, mi2 = 0;
  if (args.length == 0)                                  /* 1 */
    System.out.println("No numbers");
  else {
    mi1 = Integer.parseInt(args[0]);
    if (args.length == 1)                                /* 2 */
      System.out.println("Smallest = " + mi1);
    else {
      int obs = Integer.parseInt(args[1]);
      if (obs < mi1)                                     /* 3 */
        { mi2 = mi1; mi1 = obs; }
      for (int i = 2; i < args.length; i = i+1) {        /* 4 */
        obs = Integer.parseInt(args[i]);
        if (obs < mi1)                                   /* 5 */
          { mi2 = mi1; mi1 = obs; }
        else if (obs < mi2)                              /* 6 */
          mi2 = obs;
      }
      System.out.println("The two smallest are "
                         + mi1 + " and " + mi2);
    }
  }
}
```

Example: structural test of MinTwo

Choice	Input property	Test case id
1 true	No numbers	A
1 false	At least one number	B
2 true	Exactly one number	B
2 false	At least two numbers	C
3 false	Second number $\geq$ first number	C
3 true	Second number $<$ first number	D
4 zero times	Exactly two numbers	D
4 once	Exactly three numbers	E
4 more than once	At least four numbers	H
5 true	Third number $<$ current minimum	E
5 false	Third number $\geq$ current minimum	F
6 true	Third number $\geq$ current minimum and $<$ second least	F
6 false	Third number $\geq$ current minimum and $\geq$ second least	G

Example: structural test of MinTwo

Test case id	Input	Expected output
A		`No numbers`
B	17	`Smallest = 17`
C	27 29	`The two smallest are 27 and 29`
D	39 37	`The two smallest are 37 and 39`
E	49 48 47	`The two smallest are 47 and 48`
F	59 57 58	`The two smallest are 57 and 58`
G	67 68 69	`The two smallest are 67 and 68`
H	77 78 79 76	`The two smallest are 76 and 77`

Error!
Input data set C produces wrong results:

```
The two smallest are 27 and 0
```

The variable `mi2` is not assigned a value before it is printed.It retains its initial value, 0.

Corrected implementation (MinTwo.java)

```
public static void main (String[] args) {
  int mi1 = 0, mi2 = 0;
  if (args.length == 0)                                    /* 1 */
    System.out.println("No numbers");
  else {
    mi1 = Integer.parseInt(args[0]);
    if (args.length == 1)                                  /* 2 */
      System.out.println("Smallest = " + mi1);
    else {
      int obs = Integer.parseInt(args[1]);
      mi2 = obs;
      if (obs < mi1)                                       /* 3 */
        { mi2 = mi1; mi1 = obs; }
      for (int i = 2; i < args.length; i = i+1) {          /* 4 */
        obs = Integer.parseInt(args[i]);
        if (obs < mi1)                                     /* 5 */
          { mi2 = mi1; mi1 = obs; }
        else if (obs < mi2)                                /* 6 */
          mi2 = obs;
      }
      System.out.println("The two smallest are "
                         + mi1 + " and " + mi2);
    }
  }
}
```


Functional test

Goal: to see whether the program solves the given problem.
Method: try to show that the program *does not* solve the problem.
Test cases must cover “typical” as well as “extreme” data.

Prerequisites for functional test

1. A fairly precise description of the problem.
2. Ideas of ‘difficult’ cases and wrong ways to solve the problem.
3. The expected output data can by calculated or approximated without using the program.

Designing a functional test may reveal ambiguities in the description of the problem.
Designing a functional test may be a good way to begin developing the program.

Example: Requirements

Write a program that takes some integers as input and prints the two smallest of these, or the smallest in case there is only one.

Ambiguity: What should we do with an empty list of numbers?

Clarification: We assume that an error message `No numbers` should be given.

Example: Functional test

Table of input data properties:

Input property	Test case id
No numbers	A
One number	B
Two numbers, equal	I
Two numbers, increasing	C
Two numbers, decreasing	D
Three numbers, two equal	J
Three numbers, increasing	G
...	...

+ table with input and expected output for each test case
Note that the functional and the structural test can share some of the test cases.

Structural versus functional test

Structural and functional test complement each other.

Structural test:

- 'Mechanic', demands a systematic approach but not a deep understanding of the problem.
- Finds logical errors in the program.
- May lead to improvements of the program.

Functional test:

- Independent of the program.
- Need not be changed when the program is changed (but when the problem is changed).
- Finds unobserved/unclear requirements.
- May lead to a more precise problem description.