Introductory Programming, IMM, DTU
Systematic Software Test

Peter Sestoft[a]

- Programs often contain unintended errors — how do you find them?
- Structural test
- Functional test

Notes: *Systematic Software Test*,
`http://www.dina.kvl.dk/~sestoft/programmering/struktur.pdf`

a. Translated into English by Morten P. Lindegaard. Minor changes and some additions made by Anne Haxthausen.

Motivation

Non-trivial programs almost always contain unintended (but avoidable) errors.

Errors in programs may have severe consequences:

- In the Gulf war (1991), some Patriot missiles failed to hit incoming Iraqi Scud missiles (which subsequently killed people on the ground).
- Errors in the software controlling the baggage handling system of Denver International Airport delayed its opening by a year (1995), causing losses of around 360 million dollars.
- The first launch of the European Ariane 5 rocket failed (1996), causing losses of hundreds of million dollars. (Programming error and insufficient test).
- Errors in poorly designed control software in the Therac-25 radio-therapy equipment (1987) exposed several cancer patients to heavy doses of radiation, killing some.

Our programs are simpler and cause smaller accidents. But the programs should still not contain errors.

You may *prove* that the programs are correct (by loop invariants etc.).

This is done for e.g. Metro control-programs (France), miscellaneous equipment in spacecraft (NASA), . . .

But it is often too expensive, too labourious, or too time-consuming.

Furthermore, it only prevents certain kinds of errors. Badly designed user-interfaces are e.g. not prevented.

Software test (afprøvning)

Systematic activity.

Goal: to reveal errors in the program

Input data set: carefully designed specially for this purpose.

Systematic software test is quality assessment.

Structural test and functional test

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

Structural test is also known as internal test or ‘white-box test’.

Functional test is also known as functional test or ‘black-box test’.

In both cases, the first task is to design a *test suite* (afprøvning) containing:

- a table of input data properties
- a table of input data set and the corresponding, expected output data

To test the program, it is run with the input data sets,
after which the actual output data is compared with the expected output data.

Structural test: Conditional- and repetition statements

Weak purpose: all parts of the program have been executed.

Especially, all branches of conditional statements (`if`, `switch`, `try-catch`, ...) must have been executed.

Strong purpose: every repetition statement has been executed zero times, executed exactly one time, and executed more than one time.

This checks that variables have reasonable values before the first execution of the body of the loop, as well as the following executions.

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

Structural test: Composite logical expressions

Test all possible combinations of truth values for terms.

Conjunction (logical and, &&):

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

yields

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

Disjunction (logical or, ||):

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

yields

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

Structural test, example 1: find the smallest and the greatest number (Minmax.java)

```
public class Minmax {
  public static void main ( String[] args ) {
    int mi, ma; //current minimum and maximum
    if (args.length == 0)                                  /* 1 */
      System.out.println("No numbers");
    else {
      mi = ma = Integer.parseInt(args[0]);
      for (int i = 1; i < args.length; i = i+1) {          /* 2 */
        int obs = Integer.parseInt(args[i]);
        if (obs > ma) ma = obs;                            /* 3 */
        else if (mi < obs) mi = obs;                       /* 4 */
      }
      System.out.println("Minimum = " + mi + "; maximum = " + ma);
    }
  }
}
```


Table of test cases

Choice	Input data set	Input property
1 true	A	No numbers
1 false	B	At least one number
2 zero times	B	Exactly one number
2 once	C	Exactly two numbers
2 more than once	E	At least three numbers
3 true	C	Current number $>$ current maximum
3 false	D	Current number $\leq$ current maximum
4 true	E	Current number $\leq$ current maximum and $>$ current minimum
4 false	E	Current number $\leq$ current maximum and $\leq$ current minimum

In the program: current maximum == ma, current minimum == mi, current number == obs

Table of input data sets and expected output data

Input data set	Contents	Expected output
A		`No numbers`
B	17	`Minimum = 17; maximum = 17`
C	27 29	`Minimum = 27; maximum = 29`
D	39 37	`Minimum = 37; maximum = 39`
E	49 47 48	`Minimum = 47; maximum = 49`

Error!

Input data sets D and E yield wrong output:

```
Minimum = 39; maximum = 39
Minimum = 49; maximum = 49
```

Reason: erroneous condition at 4.

Corrected program

```
public class Minmax {
  public static void main ( String[] args ) {
    int mi, ma; //current minimum and maximum
    if (args.length == 0)                                  /* 1 */
      System.out.println("No numbers");
    else {
      mi = ma = Integer.parseInt(args[0]);
      for (int i = 1; i < args.length; i = i+1) {          /* 2 */
        int obs = Integer.parseInt(args[i]);
        if (obs > ma) ma = obs;                            /* 3 */
        else if (obs < mi) mi = obs;                       /* 4 */
      }
      System.out.println("Minimum = " + mi + "; maximum = " + ma);
    }
  }
}
```


Corrected test cases

Choice	Input data set	Input property
1 true	A	No numbers
1 false	B	At least one number
2 zero times	B	Exactly one number
2 once	C	Exactly two numbers
2 more than once	E	At least three numbers
3 true	C	Current number $>$ current maximum
3 false	D	Current number $\leq$ current maximum
4a true	E	Current number $\leq$ current maximum and $<$ current minimum
4a false	E	Current number $\leq$ current maximum and $\geq$ current minimum

The old input data sets may be used again.

Structural test, example 2: find the two smallest numbers (possibly equal) (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);
    }
  }
}
```

Table of test cases

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

Table of input data sets

Input data set	Contents	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 (in case C).

It retains its initial value, 0.

An appropriate assignment of `mi2` is necessary.

Corrected program

```
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: Does the program solve the problem?

Goal: to see whether the program solves the given problem.

Method: try to show that the program *does not* solve the problem.

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.

Functional test, example 1: find the smallest and the greatest number

Given a (possibly empty) sequence of numbers, find the greatest and the smallest of these numbers.

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

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

(What other sensible possibility is there?)

Table of input data properties

Input data set	Input property
A	No numbers
B	One number
C1	Two numbers, equal
C2	Two numbers, increasing
C3	Two numbers, decreasing
D1	Three numbers, increasing
D2	Three numbers, decreasing
D3	Three numbers, greatest in the middle
D4	Three numbers, smallest in the middle

Table of input data sets and expected output data

Input data set	Contents	Expected output
A		No numbers
B	17	`Minimum = 17; maximum = 17`
C1	27 27	`Minimum = 27; maximum = 27`
C2	35 36	`Minimum = 35; maximum = 36`
C3	46 45	`Minimum = 45; maximum = 46`
D1	53 55 57	`Minimum = 53; maximum = 57`
D2	67 65 63	`Minimum = 63; maximum = 67`
D3	73 77 75	`Minimum = 73; maximum = 77`
D4	89 83 85	`Minimum = 83; maximum = 89`

Functional test, example 2: Find the greatest difference

Given a (possibly empty) sequence of numbers, find the greatest difference between two consecutive numbers.

Ambiguity: What should we do with a sequence containing zero or one number?

Clarification: We assume that error messages should be given – `No numbers` and `Only one number`, respectively

Ambiguity: Greatest signed difference or unsigned difference?

Clarification: We assume that is should be the numeric difference (i.e. unsigned difference).

Table of input data properties

Input data set	Input property
A	No numbers
B	One number
C1	Two numbers, equal
C2	Two numbers, increasing
C3	Two numbers, decreasing
D1	Three numbers, increasing difference
D2	Three numbers, decreasing difference

Table of input data sets and expected output data

Input data set	Contents	Expected output
A		`No numbers`
B	17	`Only one number`
C1	27 27	`0`
C2	36 37	`1`
C3	48 46	`2`
D1	57 56 59	`3`
D2	69 65 67	`4`

Functional test, example 3: Legal dates

Given a day of the month day and a month mth, decide whether they determine a legal data in a non-leap year. The day and the month are given as integers.

Examples: 31/12 and 31/8 are legal dates, whereas 29/2 and 1/13 are not.

Input data set	Contents	Expected output
A	0 1	Illegal
	1 0	Illegal
	1 1	Legal
	31 1	Legal
	32 1	Illegal
	28 2	Legal
	29 2	Illegal
	31 3	Legal
	32 3	Illegal
	30 4	Legal
	31 4	Illegal
	31 5	Legal
	32 5	Illegal
	30 6	Legal
	31 6	Illegal
	31 7	Legal
	32 7	Illegal
	31 8	Legal
	32 8	Illegal
	30 9	Legal
	31 9	Illegal
	31 10	Legal
	32 10	Illegal
	30 11	Legal
	31 11	Illegal
	31 12	Legal
	32 12	Illegal
	1 13	Illegal

Structural versus functional test

Structural and functional test often use the same input data set.

Advantages of structural test

'Mechanic', demands a systematic approach but not a deep understanding of the problem.

Finds logical errors in the program.

Gives the person doing the testing (or the programmer) an opportunity to study the program in detail.

May lead to modifying the program and as a result a better program.

Covers all the details of the solution to the problem.

Advantages of functional test

Independent of the program.

Need not be changed when the program is changed (but when the problem is changed).

Gives the person doing the testing an opportunity to study the (description of the) problem in detail.

May lead to modifying the description of the problem, making it more precise.

Practical hints about rerunning tests

Rerun a test after each modification or improvement of the program.

Save the input data sets so they easily can be rerun, e.g. in a file `testminmax.bat` under Windows, e.g.:

```
java Minmax >> testminmax.res
java Minmax 17 >> testminmax.res
java Minmax 27 29 >> testminmax.res
java Minmax 39 37 >> testminmax.res
```

or in a shell script `testminmax` under Linux/Unix, e.g.:

```
#! /bin/csh
java Minmax >> testminmax.res
java Minmax 17 >> testminmax.res
java Minmax 27 29 >> testminmax.res
java Minmax 39 37 >> testminmax.res
```

The file must be on your path, and for Linux/Unix you should give it execution status with the command:

```
chmod a+x testminmax
```

Running such a script will cause output data to be saved in a textfile `testminmax.res`.

Before running the test, write expected results in a file `testminmax.exp`.

The results can be compared automatically with expected results in `testminmax.exp` (using `diff` under Unix/Linux or `fc` under MS DOS).

Testing (non main) methods of a class

So far we have considered how to test a `main` method taking arguments on the command line.

For a non main method, the principles for making a test suite (presented in two tables) are the same as for main methods.

However, to execute the test, you must write a special *test class* that has a main method that invokes the method with each input data set of the test suite.

The main method can either provide you with the results of the invokations (so that you can compare them with expected output) or even better make the comparison for you.

Sample structure of a test class

```
class X {
  static ResType m(ArgType x) { ... } //method to be tested
}

public class XTest {  // test class
  public static void main (String[] args) {
    //test the data sets of the test suite
    test(input1, expectedoutput1);
    ...
    test(inputn, expectedoutputn);
  }

  private static void test(ArgType a, ResType expected) {
    ResType res = X.m(a);

    if ( res differs from expected )
      System.out.println("m(" + a + ") returns " + res +
                         " -- differs from expected " + expected);
   }
}
```


Testing graphical user interfaces

Testing graphical user interfaces (with windows and a mouse) is cumbersome:

- One must describe carefully step by step what actions the test person must perform, and what the program's expected reactions are.
- Executing the test must be done manually; it cannot be rerun automatically.

Test in perspective

- Testing can improve our confidence in a program,
  but it can never *prove* that a program has no errors.
- The saying of the statistician is relevant: *Absence of evidence isn't evidence of absence!*
- *The tester* thinks that the test is successful if it does find errors.
- *The programmer* thinks that the test is successful if it does *not* find errors.
- When tester and programmer is the same person, there is a psychological conflict.
- It takes time to design a test. This motivates for
  - avoiding superfluous choice- and repetition statements (simplifies the structural test);
  - avoiding superfluous special cases in the description of the problem (simplifies the functional test).
- Programs must be tested
  - if errors can damage humans or animals;
  - if errors can lead to considerable economic losses;
  - if they are used to draw scientific conclusions.
- It is out of scope of this course to describe all aspects of testing.