

Introduction to Programming – Concepts and Tools

Carsten Butz
IT University of Copenhagen

Week 8

Today's Goals

- Feedback on assignments
- Short note about project period
- Summary of course so far
- Lecture

- Testing (slides)
- Exception handling (blackboard)

Relevant sources for the first two topics:

Stevens/Pooley: Using UML, Addison–Wesley 2000.

(You are not supposed to get a copy of that source.)

Peter Sestoft: Systematic software test, Notes 1998.

So far:

- Procedural programming
 - Syntactical components, number types
 - Control structures, loops
 - Methods (procedures), method overloading, recursion
 - Arrays
 - Searching and sorting of arrays
- Fundamentals of object oriented programming
 - Classes, objects, constructors, access
 - Inheritance, class hierarchies

So far:

- Concepts of procedural programming
 - Data types
 - Control flow: loops, if–then–else, switch
 - Data structures: arrays
 - Methods
 - Call by value, call by name
 - Searching, sorting, analysis of algorithms
 - ...
- Concepts of OOP
 - Classes, objects
 - Encapsulation, inheritance, polymorphism
 - UML (Unified modelling language)
- Syntax of the programming language Java

Software Engineering

Modern software is

- highly complex
- in use in safety critical systems
- thus engineered, not just programmed.

Aspects of software engineering are discussed in courses like

- Object Oriented Programming
- Advanced Object Oriented Programming
- Project Management (IT–, Software–, ...)

Techniques are also applied during the project periods and during the writing of your thesis (speciale).

Software Engineering

Good systems are

- useful, usable
- reliable, robust
- flexible
- affordable
- available.

Problem: Lots of failures, even drastic ones.

Testing: Errors

Error types:

- Syntax errors (compiler)
- Type errors (compiler)
- Semantic errors

Testing: Types

Structural Testing (white-box/internal testing)

- Focuses on the code
- Test suite that shows all branches of if, while, switch are executed (correctly)

Functional Testing (black-box testing)

- Focuses on the problem
- Test suite uses typical and extreme data

The two types of tests are complementary to each other!

Structural Test: Example 1

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	Number > current max
3 false	E	Number <= current max
4 true	E number 3	Number <= current max and > current min
4 false	E number 2	Number <= current max and <= current min

Input data set	Contents	Expected output
A	(no numbers)	No numbers
B	17	17 17
C	27 29	27 29
D	39 37	37 39
E	49 47 48	47 49

Structural Test: Example 1

Running the program shows that the outputs are wrong (they disagree with the expected outputs) for input data sets D and E.

Leads to program's choice 4 is wrong, it should be

```
else if (obs < mi) mi = obs /* 4a */
```

New structural test:

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	Number > current max
3 false	E	Number <= current max
4a true	E number 2	Number <= current max and < current min
4a false	E number 3	Number <= current max and <= current min

Structural Testing: Summary

Program statements should be tested as follows:

Statement	Cases to test
If	True/false
For	zero/one/more than one iteration
While	zero/one/more than one iteration
Do-while	one/more than one iteration
Switch	Every branch must be executed

Composite logical expressions such as

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

must be tested for all possible combinations of truth values.

Functional Test: Example 1

Same problem as above, this time no specific program is given, or the actual implementation is not known.

Suggested tests:

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



Functional Test: Example 1

The choice of input data sets may be critical (in case C2 it might make a difference whether we test with 35 37 or with 0 37)!

Input data set	Input property	Expected output
A	(no numbers)	Error message
B	17	17 17
C1	27 27	27 27
C2	35 36	35 36
C3	46 45	45 46
D1	53 55 57	43 47
D2	67 65 63	63 67
D3	73 77 75	73 77
D4	89 83 85	83 89



Practical Hints

- Avoid testing where the expected output is 0 (Java).
- In languages like C++ variables are not initialized, but the memory address contains some value which might be the expected outcome (accidentally).
- Automate tests so that they can be rerun after modifying the program
- Larger packages or programs with user interfaces require more complex tests. The actual tests (which buttons are pressed in which order, ...) must be documented carefully.



Tests in Perspective

- Tests can never prove correctness, but give confidence.
- Structural tests are often easier to design than functional tests.
- A test is successful if it does find an error (the programmer might think differently). It is best if some of the tests are not designed by the actual programmer of the tested module.
- Designing good and complete test suits takes much effort and time.
- Tests (design, what is tested, why is it tested, expected outcome, observed outcome, evaluation) are often part of software documentation.