# Software Development II
# Unit 3: *The idea of testing*

Markus Roggenbach

February 2019

# You will learn

How to establish conformance between a

- specification
- program

via testing.

Complete testing is impossible.

Therefore we test only selected cases.



# Testing

# Establishing a program's quality

Aim:

System under Test (SUT)
conforms to a
Specification

In our context:

| | | |
|---|---|---|
| SUT | – | Java method / Java class |
| Specification | – | Computational Problem |

Activity: Systematically experiment with the SUT.



# A sample test cycle

# Example: Computational Problem

**Multiplication:**

**Input:** natural numbers $a$, $b$

**Output:** the natural number $a * b$

# Example: SUT

```
public static int myMult(int a, int b) {
    int d = 0;
    int i = 0;
    while (i <= a) {
        d = d + b;
        i++;
    }
    return d;
}
```

Note: `myMult` has a bug.

# Test case

Description of an experiment in the physical world.

**In our context:**

A test case consists of

- a name,
- the input values, and
- the expected result.

A test suite is a collection of test cases.

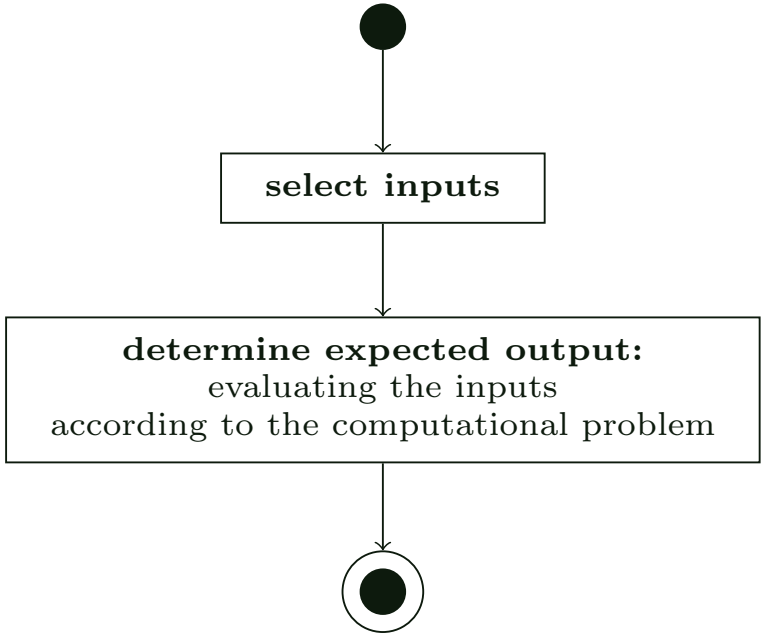
# Example: Test suite for the Multiplication problem

| name | input 1 | input 2 | expected result |
|---|---|---|---|
| T1 | 1 | 2 | 2 |
| T2 | 0 | 0 | 0 |
| T3 | -1 | 3 | don't know |
| T4 | 4 | -2 | don't know |
| T5 | -6 | -7 | don't know |

(here – input 1 and input 2 are some randomly chosen numbers; later we will discuss methods to systematically select inputs)



# The process of designing a test suite




Shown here as activity diagram from the Unified Modelling Language (UML).

# Test execution

Performing the (physical) experiments as described by the test cases.

In our context: execute the SUT for each test case, i.e.,

- provide the inputs for the SUT
- fill in a column "actual result" of the previous table

No need to execute test cases with result "don't know".

# Example: The code for calling the method unknown

```
// read in input-1
System.out.print("input-1: ");
in1 = input.nextInt();

// read in input-2
System.out.print("input-2: ");
in2 = input.nextInt();

// printing the result
System.out.println( "actual result = " + unknown(in1,in2));
```

# Example: filling in the actual result

| name | input 1 | input 2 | expected result | actual result |
|---|---|---|---|---|
| T1 | 1 | 2 | 2 | |
| T2 | 0 | 0 | 0 | |
| T3 | -1 | 3 | don't know | |
| T4 | 4 | -2 | don't know | |
| T5 | -6 | -7 | don't know | |

# Test evaluation

Determine if the SUT passed or failed the test suite.

To this end, compare for each test case

- the actual result with the expected result and
- give the verdict “pass” if they are “the same”, otherwise give the verdict “fail”

We say that a SUT passes a test suite if for all test cases it obtained the verdict “pass”.

# Example: Test evaluation

| name | input 1 | input 2 | expected result | actual result | quality |
|---|---|---|---|---|---|
| T1 | 1 | 2 | 2 | | |
| T2 | 0 | 0 | 0 | | |
| T3 | -1 | 3 | don't know | | |
| T4 | 4 | -2 | don't know | | |
| T5 | -6 | -7 | don't know | | |

# Test documentation

files

- evidence that test execution has happened
- the actual results (e.g., in form of log-files)
- test evaluation

Is part of the "safety-case" when one asks for certification; plays important role in case of legal disputes.

# Why the design of test suites is important

# Complete testing is impossible

```
The values of the integral types are integers
in the following ranges:
  For int, from -2147483648 to 2147483647, inclusive.
```

Java int has roughly $5 * 10^9$ values

Number of combinations of two int values: $2.5 * 10^{19}$.

Assumption: $10^6$ Test cases per second.

Yields for total testing:

$2.5 * 10^{19}/10^6 sec = 2.5 * 10^{13}\ sec \sim 2.5 * 10^{11}\ min \sim$

$2.5 * 10^9\ h \sim 2.5 * 10^7\ days \sim$

$2.5 * 10^4\ years$



# Focus of the module: Design of Test-Suites

Main approaches for "rules of thumb":

Black box testing:

Test case selection based on specification.

White box testing:

Test case selection based on program code.



# Importance of testing in industry

# Effort distribution, industry, 2005

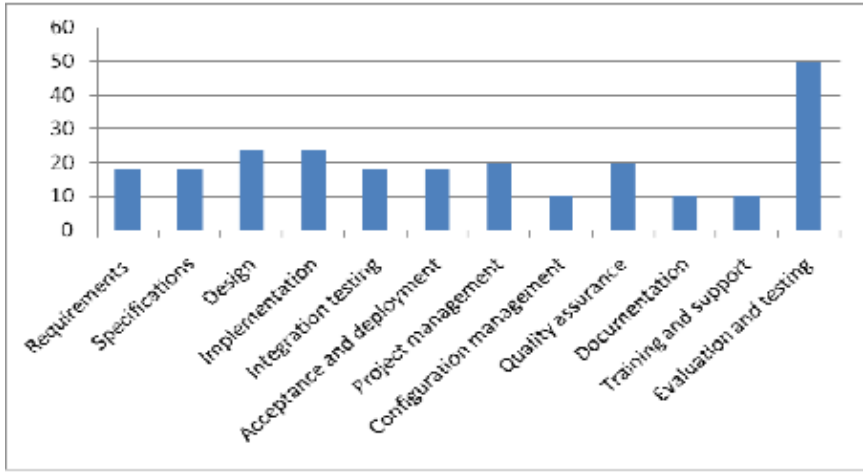
Based on studies reported in the general industry literature, the distribution of effort across the software development life cycle is typically along the lines of the following:

- Requirements: 15-20 percent
- Analysis and Design: 15-20 percent
- Construction (code and unit testing): 25-30 percent
- System Testing: 15-20 percent
- Implementation: 5-10 percent.

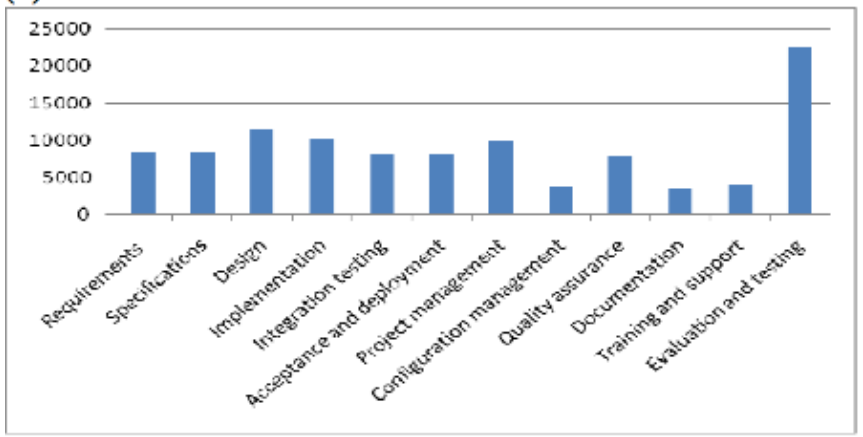
`http://it.toolbox.com/blogs/enterprise-solutions/effort-distribution-across-the-software-lifecycle-6304`



# Saleh 2011




(a) Effort distribution




(b) cost distribution

K Saleh: Effort and Cost Allocation, Intern. J. of Computers, Vol 5(1), 2011.

# What you have learned today

# Definitions

- Test case
- Test suite
- Test execution
- Test evaluation

# You should be able to explain by example

- The process of how to design a test suite.
- Why complete testing is not feasible.