Course-Software-Testing

MarDATA

Enno Prigge Willi Rath Dilip Hiremath Sven Gundlach

Kiel University

18th November 2020

Structure

1. Introduction
2. Whitebox & Blackbox Testing
3. Mocking
4. Unit Test
5. Integration Testing

Structure

1. Introduction
2. Whitebox & Blackbox Testing
3. Mocking
4. Unit Test
5. Integration Testing

What is software testing?

Testing:

- Testing is the process of executing a program with the intent of finding errors (Myers, G.J., 1979. The Art of Software Testing John Wiley and Sons Ltd.)
- Software testing is the process of exercising or evaluating a system or system component by manual or automated means to verify that it satisfies specified requirements. (IEEE 83a)

What is software testing?

Testing:

- Testing is the process of executing a program with the intent of finding errors (Myers, G.J., 1979. The Art of Software Testing John Wiley and Sons Ltd.)
- Software testing is the process of exercising or evaluating a system or system component by manual or automated means to verify that it satisfies specified requirements. (IEEE 83a)

Principle of Testing:

Testing can show presence of bugs but not absence or a proof of correctness. No bug found shows not a correct system.

New features: „Naive“ Workflow

Typical „unstructured “ approach:

- Implementation of new features
- Call and test of features from main method, output with **System.out.println**
- Manual check whether results are correct (or at least plausible)
- Test code in Main method is not used later

New features: „Naive“ Workflow

Typical „unstructured “ approach:

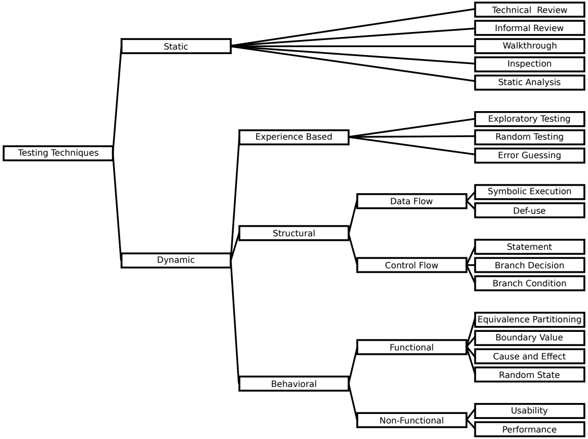
- Implementation of new features
- Call and test of features from main method, output with **System.out.println**
- Manual check whether results are correct (or at least plausible)
- Test code in Main method is not used later

Cons?

- Code in Main method not „belonging there“
- Test code is discarded, although it is still useful later
- Tests are no longer executed

→ Regressions are not recognized

Testing Techniques



- Testing Techniques
  - Static
    - Technical Review
    - Informal Review
    - Walkthrough
    - Inspection
    - Static Analysis
  - Dynamic
    - Experience Based
      - Exploratory Testing
      - Random Testing
      - Error Guessing
    - Structural
      - Data Flow
        - Symbolic Execution
        - Def-use
      - Control Flow
        - Statement
        - Branch Decision
        - Branch Condition
    - Behavioral
      - Functional
        - Equivalence Partitioning
        - Boundary Value
        - Cause and Effect
        - Random State
      - Non-Functional
        - Usability
        - Performance

Testing Approaches

- Level of knowledge and access to implementation
  - Whitebox Testing
  - Blackbox Testing

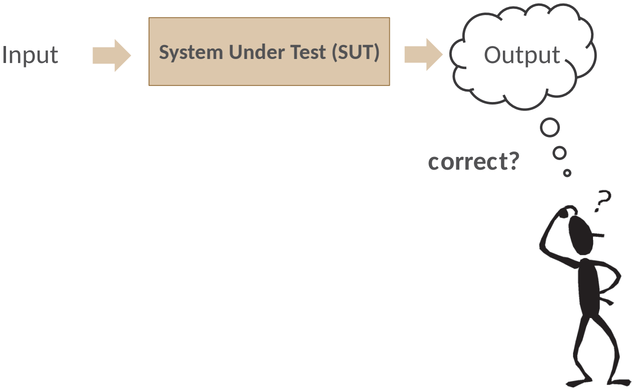
Testing Approaches

- Level of knowledge and access to implementation
  - Whitebox Testing
  - Blackbox Testing
- Testing scope
  - System testing
  - Module testing
  - Integration testing
  - Unit testing

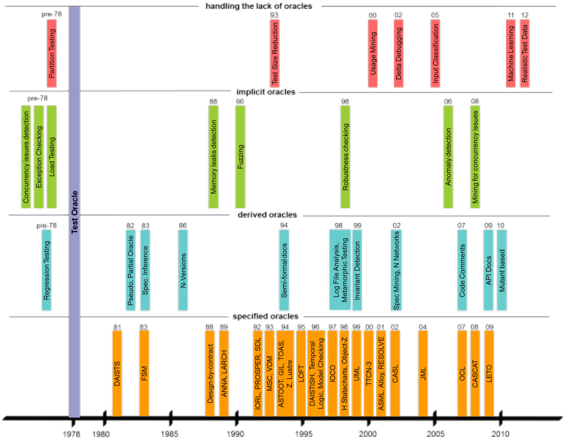
Testing Approaches

- Level of knowledge and access to implementation
  - Whitebox Testing
  - Blackbox Testing
- Testing scope
  - System testing
  - Module testing
  - Integration testing
  - Unit testing
- Concepts
  - Regression Testing
  - Fuzzing
  - Metamorphic testing
  - Design by Contract
  - Machine Learning
  - . . .

Test Oracle



Concepts by type of Test Oracle



Harman, Mark, et al. "A comprehensive survey of trends in oracles for software testing." University of Sheffield, Tech. Rep. CS-13-01 (2013).

Test driven Development:

Software development approach in which a test is written before writing the code

Definitions of terms

Test Condition: What needs to be tested. Item or event that could be verified.

Test case: Testing test condition or objectives, consist of input values, preconditions, expected results, post conditions.

Test procedure: Collection of test cases arranged in sensible order to execute a test.

Test suite: Entire test.

Test data: Input data for a test case.

Expected result: Outputs, changes to data and states and other consequences anticipated to happen.

Test script: Automated test procedure.

Definitions of terms

Test Condition: What needs to be tested. Item or event that could be verified.

Test case: Testing test condition or objectives, consist of input values, preconditions, expected results, post conditions.

Test procedure: Collection of test cases arranged in sensible order to execute a test.

Test suite: Entire test.

Test data: Input data for a test case.

Expected result: Outputs, changes to data and states and other consequences anticipated to happen.

Test script: Automated test procedure.

Definitions of terms

Test Condition: What needs to be tested. Item or event that could be verified.

Test case: Testing test condition or objectives, consist of input values, preconditions, expected results, post conditions.

Test procedure: Collection of test cases arranged in sensible order to execute a test.

Test suite: Entire test.

Test data: Input data for a test case.

Expected result: Outputs, changes to data and states and other consequences anticipated to happen.

Test script: Automated test procedure.

Definitions of terms

Test Condition: What needs to be tested. Item or event that could be verified.

Test case: Testing test condition or objectives, consist of input values, preconditions, expected results, post conditions.

Test procedure: Collection of test cases arranged in sensible order to execute a test.

Test suite: Entire test.

Test data: Input data for a test case.

Expected result: Outputs, changes to data and states and other consequences anticipated to happen.

Test script: Automated test procedure.

Definitions of terms

Test Condition: What needs to be tested. Item or event that could be verified.

Test case: Testing test condition or objectives, consist of input values, preconditions, expected results, post conditions.

Test procedure: Collection of test cases arranged in sensible order to execute a test.

Test suite: Entire test.

Test data: Input data for a test case.

Expected result: Outputs, changes to data and states and other consequences anticipated to happen.

Test script: Automated test procedure.

Definitions of terms

Test Condition: What needs to be tested. Item or event that could be verified.

Test case: Testing test condition or objectives, consist of input values, preconditions, expected results, post conditions.

Test procedure: Collection of test cases arranged in sensible order to execute a test.

Test suite: Entire test.

Test data: Input data for a test case.

Expected result: Outputs, changes to data and states and other consequences anticipated to happen.

Test script: Automated test procedure.

Definitions of terms

Test Condition: What needs to be tested. Item or event that could be verified.

Test case: Testing test condition or objectives, consist of input values, preconditions, expected results, post conditions.

Test procedure: Collection of test cases arranged in sensible order to execute a test.

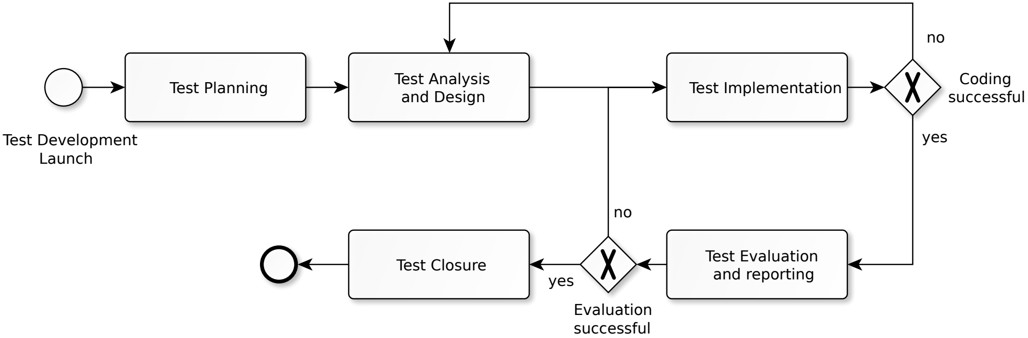
Test suite: Entire test.

Test data: Input data for a test case.

Expected result: Outputs, changes to data and states and other consequences anticipated to happen.

Test script: Automated test procedure.

Test driven development process



Risk mitigation

How much testing is enough?

Exhaustive test: Test approach in which the test suite comprises all combinations of input values and preconditions.

- Not feasible except for trivial cases

Risk mitigation: Use risks and priorities to focus testing efforts.

- Impossible to understand all end user scenarios
- Impossible to analyze all implications of code changes

Automation: Automated testing can have significant coverage

Equivalence Class Partitioning

Aim

Divide input parameter and output ranges into equivalence classes.

Assumption

When processing a representative from an equivalence class, a program behaves the same way as with all other values from this equivalence class.

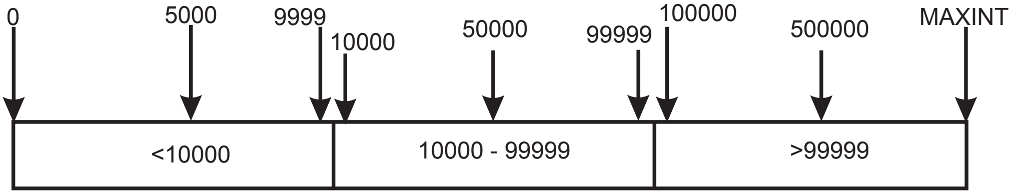
Representative value for a test case

Any representative value from each class.

Boundary Value Analysis

- Test cases that cover the boundary values of equivalence classes or that are in the immediate vicinity of the boundaries uncover errors very often.
- Not any element from the equivalence class is valid as representative for all boundaries.
- Pick one or more elements such that every boundary of the equivalence class is tested.

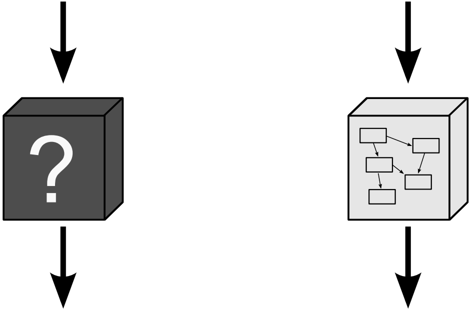
Equivalence classes with boundary values



Input with three equivalence classes.

Structure

1. Introduction
2. Whitebox & Blackbox Testing
3. Mocking
4. Unit Test
5. Integration Testing



known input

known ouput

Blackbox Test

known input

known ouput

Whitebox Test

Whitebox Testing of Modules

testing in the small

Statement Coverage: **Statement coverage criterion**

All executable statements in the source code are executed at least once.

Edge coverage: **Edge-coverage criterion**

Each edge of the control graph being traversed at least once.

Path (Branch) Coverage: **Path-coverage criterion**

All paths leading from the initial to the final node of the control graph being traversed.

(Compound) Condition Coverage: **Condition coverage criterion**

Each possible combination of conditions must be executed at least once.

Blackbox Testing of modules

testing in the large

Idea: A part of a program is tested without knowledge of the internal implementation.

Problem: How to define the test set?

Defining the test set can only rely on the specification.

Formal specifications are then a prerequisite for the systematic generation of test sets.

Blackbox Testing of modules (contd.)

Method: Derive test cases from the program specification.

Disregard program structure.

Pro: Comprehensive but low-redundancy testing of the specified functionality.

The key factor is the functional coverage.

Con: Oversight of functionality.

Defining a test case:

- Functional equivalence class formation (equivalence class partitioning)
- Boundary value analysis (BVA)
- Special value testing

Structure

1 Introduction

2 Whitebox & Blackbox Testing

3 Mocking

4 Unit Test

5 Integration Testing

Mock objects: Categories

Dummy objects are passed around but never actually used. Usually they are just used to fill parameter lists.

Fake objects actually have working implementations, but usually take some shortcut which makes them not suitable for production (an in-memory database is a good example).

Stubs provide canned answers to calls made during the test, usually not responding at all to anything outside what's programmed in for the test.

Spies are stubs that also record some information based on how they were called.

Mocks are [. . .] objects pre-programmed with expectations which form a specification of the calls they are expected to receive.

Mocking: Summary

Motivation: Replace complex objects for test:

- Simulation
- Replace with dummies
- Collect additional debug information

Connection to Unit-Testing: Extends assertions by Verification. Testing of interactions with mock objects.

Verification: Mock objects log function calls. This allows to check properties of the *stack trace (call history)* of mock objects. No formal verification

Unit Tests *vs.* Mock-Verification? Addition, not replacement but different aspects.

Structure

1. Introduction
2. Whitebox & Blackbox Testing
3. Mocking
4. Unit Test
5. Integration Testing

Unit Test

Automatic unit tests:

- Test code in a specially designated area of the project
- Execution like Main methods but tool support for:
  - Comparison with expected behavior
  - automatic execution of all tests
  - Statistics on completed and failed tests

Unit Test

Automatic unit tests:

- Test code in a specially designated area of the project
- Execution like Main methods but tool support for:
  - Comparison with expected behavior
  - automatic execution of all tests
  - Statistics on completed and failed tests

Pros:

- Separation of functionality and tests
- Tests are retained
- Tests can be run continuously, automatically
  → Regressions are identified
- Tests as quality gateway for integration of new code into the master branch or master repository

An Demo-function

```
package com.example.project;

public class Calculator {
        public int add(int a, int b) {
                return a - b;
        }
}
```

Take a look at this in eclipse.

An Demo-function

```
package com.example.project;

import static org.junit.jupiter.api.Assertions.assertEquals;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;

class MyTest {
        @Test
        @DisplayName("1␣+␣1␣=␣2")
        void addsTwoNumbers() {
            Calculator calculator = new Calculator();
            assertEquals(2, calculator.add(1, 1), "1␣+␣1␣should␣equal␣2");
        }
}
```

https://github.com/junit-team/junit5-samples/

Unit Test: Summary

Evaluation:

- Automatic tests cause few additional work!
- Help to recognize quality problems early.

Unit Test: Summary

Evaluation:

- Automatic tests cause few additional work!
- Help to recognize quality problems early.

Questions:

- How many tests should I write?
- Which parts of the software should be tested?
- And what sample data?
- From where do I know the „expected“ results?

Structure

1. Introduction
2. Whitebox & Blackbox Testing
3. Mocking
4. Unit Test
5. Integration Testing

Integration testing

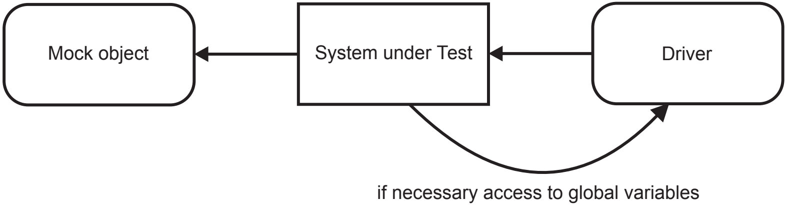
testing in the large

- Testing individual modules separately does not guarantee a correct inter-operation of multiple modules.
- Many modules cannot be tested in isolation.
- Additionally: Steps for testing complex systems (modular testing):
  - Modular tests (a.k.a. Unit Tests & Components Tests)
    Testing a component without context.
  - Incremental integration test of multiple components in their context.
  - System test: Test of the whole system in the application environment (end-to-end).
- Acceptance tests (a.k.a. Funktional tests)
  Focus on testing of 'cross cutting' functionality.

Integration testing (contd.)

- For embedded real time systems, the interoperation of hardware and software needs to be tested.
    - Digital Twins
- Incremental testing should be preferred over ‚Big-Bang'testing, which directly goes for the full system right after unit testing.
- A clear separation of interface and implementation makes integration testing easy.
    - Easier swapping of ‚mock-ups' for ‚real' modules. See, e.g., Mockito `https://github.com/mockito/`

Incremental testing



"‘Test framework"’

- Incremental tests can be bottom-up, top-down (or even jo-jo), with respect to the hierarchy of composition.
- A hierarchical architecture is very effective for this purpose.

top-down: Highest level module are tested and integrated first. Data flow is also tested early in the process.

- Pro: Limits driver
- Con: Complicate as of stubs, low level units are tested late, no early release of functionality

bottom-up: Lowest level units are tested and integrated first.

- Pro: Minimize need for stubs
- Con: Complicate as of drivers, high level logic and data flow is tested late, no early release of functionality

Jo-Jo: Testing along functional data and control flow path

- output functions are integrated in top-down
- input functions are integrated in bottom-up
- Pro: limited early release of functionality, minimize stubs and drivers

Structure

1. Introduction
2. Whitebox & Blackbox Testing
3. Mocking
4. Unit Test
5. Integration Testing

Metamorphic Testing(MT)

- comparing outputs of successive runs with varying(morphed) input data for validation
- Metamorphic Relation(MR): intrinsic function of software identified for validating

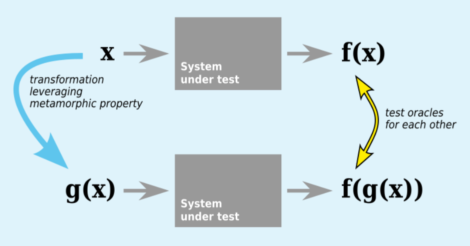

x
System under test
f(x)
transformation leveraging metamorphic property
test oracles for each other
g(x)
System under test
f(g(x))

image:Metamorphic Testing by Teemu Kanstrén from Towards Data Science

Metamorphic Relations

- MRs are usually identified based on the system under test by domain experts
- Permutative(change the order of keywords in a search)
- Additive, Multiplicative (add or multiplying by a constant, resulting in increase/no effect on output)
- Invertive(changing the signs of each data point, resulting in negative of previous output)
- Inclusive(adding a new constraint in the search criteria)
- Exclusive(removing elements from the data points)
- periodicity(explored in Hands on session)
- could be linear or quadratic

$$\cos(2x) = 2\cos^2(x)$$