MTAT.03.159: Software Testing

Lecture 01:
Introduction to Software Testing
(Textbook Ch. 1-3)

Spring 2018

Dietmar Pfahl
email: dietmar.pfahl@ut.ee
Structure of Lecture 1

• Introduction and Motivation
• Course Information
• Definitions and Principles
• Lab 1
2012: Knight Capital loses 440M USD

- August 12th: New Trading Software installed
- Administrator forgets to deploy on one out of eight server nodes
- New code repurposed a flag previously used for testing scenarios
- On that one server node, old trading algorithm interprets flag differently and starts buying and selling 100 different stocks randomly without human verification
- NYSE has to suspend trade of several stocks
- Knight Capital loses 440 Mio USD in only 30 minutes, until system is suspended
- Investors have to raise 400 Mio USD in order to rescue the company
Recall Exercise: A Pen

• Quality?

• Testing?
Software Quality – Definition

• **Software quality is the degree of conformance to explicit or implicit requirements and expectations**

Explanation:
• *Explicit*: clearly defined and documented
• *Implicit*: not clearly defined and documented but indirectly suggested
• *Requirements*: business/product/software requirements
• *Expectations*: mainly end-user expectations
Software Product Quality Model – ISO 25010 Standard
Software Quality Assurance (SQA)

- SQA is a set of activities for ensuring quality in software engineering processes (that ultimately result in quality in software products).

It includes the following activities:
- Process definition
- Process implementation
- Auditing
- Training

Processes could be:
- Software Development Methodology
- Project Management
- Configuration Management
- Requirements Development/Management
- Estimation
- Software Design
- Testing
- …
Software Quality Control (SQC)

- SQC is a set of activities for ensuring quality in software products.

It includes the following activities:
- Reviews
- Testing

(Static) Testing:
- Requirement Review
- Design Review
- Code Review
- Deployment Plan Review
- Test Plan Review
- Test Cases Review

(Dynamic) Testing:
- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing
What is Software Testing? (Static & Dynamic)

Confirm quality (**pass-test**) vs. Find defects (**fail-test**)
What is Software Testing?

The process of evaluating a program or a system

Verifying

Validating
Verification vs. Validation

Source: SEI at CMU, Donald Firesmith
Verification

Definition
• The process of evaluating work-products (not the actual final product) of a development phase to determine whether they meet the specified requirements for that phase.

Objective
• To ensure that the product is being built according to the requirements and design specifications. In other words, to ensure that work products meet their specified requirements.

Question
• Are we building the product right?
Validation

Definition

• The process of evaluating software during or at the end of the development process to determine whether it satisfies specified (or implicit) business requirements.

Objective

• To ensure that the product actually meets the user’s needs, and that the requirements were correct in the first place. In other words, to demonstrate that the product fulfills its intended use when placed in its intended environment.

Question

• Are we building the **right** product?
Testing in the Software Development Lifecycle (SDLC)

- Requ. Spec.
- Design
- Code
- Unit test
- Integration test
- System test
- Acceptance test

See Fig 1.6 Textbook
Software Testing Life Cycle (STLC)

- The STLC defines the steps/stages/phases in testing of software.
STLC integrated with SDLC
Test Techniques

Keyword testing
Exploratory testing

Load testing
Stress-testing
Black box security testing

Input space partitioning
Boundary values
Error guessing
All combinations/pairwise/$n$-wise
Coverage criterions
Mutation testing
Agile methods increase testing

[http://www.extremeprogramming.org]
Test Complexity – Quiz

Example:
• 30 variables, 2 levels
• Test all combinations

How long does it take to test, if 5 tests/sec can be executed automatically?

Answer choices:
1. Less than 10 sec
2. Less than 1 min
3. Less than 1 hour
4. Less than 1 day
5. Less than 1 year
6. More than 1 year
Test Complexity – Quiz

For 30 variables with 2 possible inputs for each, how long does it take to test all possible combinations, if 5 tests/sec can be executed automatically? Total Votes 15

<table>
<thead>
<tr>
<th>COUNT</th>
<th>PERCENT COUNTRY</th>
<th>PERCENT OVERALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Less than 10 sec</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>2: Less than 1 min</td>
<td>3</td>
<td>20%</td>
</tr>
<tr>
<td>3: Less than 1 hour</td>
<td>1</td>
<td>6.67%</td>
</tr>
<tr>
<td>4: Less than 1 day</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>5: Less than 1 year</td>
<td>3</td>
<td>20%</td>
</tr>
<tr>
<td>6: More than 1 year</td>
<td>8</td>
<td>53.33%</td>
</tr>
</tbody>
</table>
Test Complexity

Example:

- 30 variables, 2 levels
  \[ \Rightarrow 2^{30} \approx 10^9 \]
  combinations to test

- 5 tests/second \[ \Rightarrow 214748364.8 \text{ sec or} \]
  \[ \textcolor{red}{6.8 \text{ years of testing!}} \]
Structure of Lecture 1

- Introduction and Motivation
- Course Information
- Definitions and Principles
- Lab 1
Course Information/Overview

• Level: Course at bachelor's level (in English), 2\textsuperscript{nd} year
• Credits: 3 ECTS
• Prerequisite:
  – Compulsory: MTAT.03.094/LTAT.05.008 Software Engineering (6 ECTS)
  – Recommended: MTAT.03.130 Object-oriented Programming (6 ECTS)
• Work load:
  – Lectures (incl. practical work): 30 person-hours – incl. lab and exam sessions
  – Independent work (outside classroom): 48 person-hours
• Assessment:
  – 7 Homework Assignments (work in pairs) – 70\% of grade (~6 ph per lab = 42 ph)
  – Exam (written) – 30\% of grade (~36 ph)

• Grade scale: A (90\%+), B(80\%+), C(70\%+), D(60\%+), E(50\%+), F
Course Objectives

• The course
  – addresses the essential concepts of software quality control and testing
  – introduces various testing strategies and types of testing

• … and will also give an overview of
  – different software defects
  – software defect management
  – organizational aspects of software testing
Letter Grades

- **A** - An excellent performance, clearly outstanding. The candidate demonstrates excellent judgement and a high degree of independent thinking.
- **B** - A very good performance. The candidate demonstrates sound judgement and a very good degree of independent thinking.
- **C** - A good performance in most areas. The candidate demonstrates a reasonable degree of judgement and independent thinking in the most important areas.
- **D** - A satisfactory performance, but with significant shortcomings. The candidate demonstrates a limited degree of judgement and independent thinking.
- **E** - A performance that meets the minimum criteria, but no more. The candidate demonstrates a very limited degree of judgement and independent thinking.
- **F** - A performance that does not meet the minimum academic criteria. The candidate demonstrates an absence of both judgement and independent thinking.

ECTS recommended distribution:

A: 10% B: 25% C: 30% D: 25% E: 10%

ECTS = European Credit Transfer and Accumulation System
Lectures (J. Liivi 2-111)

Theory + discussions
Cover the basic aspects of software testing

• Lecture 1 (15.02) - Introduction to Software Testing
• Lecture 2 (01.03) - Black-Box Testing Techniques
• Lecture 3 (15.03) - White-Box Testing Techniques
• Lecture 4 (29.03) – Mutation Testing, Static Analysis & Defect Estimation
• Lecture 5 (12.04) - Lifecycle, Levels, Tools
• Lecture 6 (26.04) - Documentation, Organisation & Improvement
• Lecture 7 (10.05) – Industry Guest Lecture (option), Exam preparation

Overview
Technical
Technical / Managerial
# Lectures & Labs Schedule

<table>
<thead>
<tr>
<th>W24</th>
<th>W25</th>
<th>W26</th>
<th>W27</th>
<th>W28</th>
<th>W29</th>
<th>W30</th>
<th>W31</th>
<th>W32</th>
<th>W33</th>
<th>W34</th>
<th>W35</th>
<th>W36</th>
<th>W37</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lec 1</td>
<td>Lec 2</td>
<td>Lec 3</td>
<td>Lec 4</td>
<td>Lec 5</td>
<td>Lec 6</td>
<td>Lec 7</td>
<td>Lab 1</td>
<td>Lab 2</td>
<td>Lab 3</td>
<td>Lab 4</td>
<td>Lab 5</td>
<td>Lab 6</td>
<td>Lab 7</td>
</tr>
</tbody>
</table>

- Submit HW 1
- Submit HW 2
- Submit HW 3
- Submit HW 4
- Submit HW 5
- Submit HW 6
- Submit HW 7

Lab Groups on May 1 are moved to May 7
Lab Sessions (J. Liivi 2-... various rooms)

Preparation, Execution, Report – **Work in Pairs**

**New**
1. Debugging (10%)
2. Black-box & White-box testing (10%)
**New**
3. Combinatorial testing (10%)
4. Mutation testing (10%)
**New**
5. Automated Web-application testing (10%)
6. Static code analysis (10%)
7. Document inspection and defect estimation (10%)

Send reports via submission button on course wiki before your next lab starts. Only PDF files will be accepted.
GO TO LABS !!!!!

(if you don’t, you will lose marks)
Final Exam

Written exam (30%)
  • Based on textbook, lectures and lab sessions
  • Open book & open laptop

Dates:
  • Exam 1: 25-May-2015 14:15-16:15 (Liivi 2-405; limit: 49 stud.)
  • Exam 2: 28-May-2015 14:15-16:15 (Liivi 2-405; limit: 49 stud.)
  + re-take exam in June
Literature

- Textbook:
  - Ilene Burnstein: Practical Software Testing

- Additional reading materials posted on course web
Software Testing 2017/18 spring

Software Testing

Course Responsible / Instructor: Dietmar Pfahl (dietmar.pfahl at ut dot ee) - room: 302

TA: Ezequiel Scott (ezequielscott at gmail dot com)
TA: Margus Luik (mar6luik at gmail dot com)
TA: Jaanus Jaggo (jaanus.jaggo at ut dot ee)

Lectures (begin in week 24 of the academic year, on 15-Feb-2018):
- Thursday 10:15 - 12:00, J. Liivi 2-111

Labs (practice learning; begin in week 25):
- Group 1: Tuesday 14.15 - 16.00, r224 - week 25, 27, 29, 31, 33, 36, 37 - Margus
- Group 2: Tuesday 14.15 - 16.00, r207 - week 25, 27, 29, 31, 33, 36, 37 - Jaanus
- Group 5: Wednesday 14.15 - 16.00, r206 - week 25, 27, 29, 31, 33, 35, 37 - Ezequiel

Note: The lab sessions for lab groups 1 and 2 cannot take place in week 35 as they would fall on May 1 which is a national holiday. Therefore the lab sessions of May 1 are moved by one week into week 36 (marked in red colour).
SIGN UP TO MESSAGE BOARD (Slack)

(if you don’t, you will miss up to date info)
Structure of Lecture 1

• Introduction and Motivation
• Course Information
• Definitions and Principles
• Lab 1
Definition 1: Error – Fault – Failure
(according to IEEE Standard & textbook)

• **Failure** is an event caused by a **fault**, and a **fault** is an anomaly of the software caused by an **error**
  • **Error** – mistake made by human (e.g., programmer)
  • **Fault** – wrong/missing statement in the software (code)
  • **Failure** – inability to perform the program’s required functions (correctly)
  • Defect ? – Bug ?

• **Debugging** / Fault localization – localizing, repairing, re-testing.
Origins and Impact of Faults

Fault sources
- Lack of skills/training
- Oversight
- Poor communication
- ‘Lost in translation’
- Immature process

Fault context
- Impact on/of software program
  - Errors
    - Faults
      - Failures

User’s point of view
- Poor quality software
- User dissatisfaction

Fig 3.1
Definition 2: Error – Fault – Failure
(as it is often used in IDEs/tools)

- **Failure** is an event caused by an **error**, **error** is a state of the program caused by a **fault** in the code
  - **Fault** – wrong/missing statement in code (resulting in error)
  - **Error** – incorrect program state (may result in a failure)
  - **Failure** – inability to perform its required functions (correctly)
  - Defect ? – Bug ?

- **Debugging** / Fault localization – localizing, repairing, re-testing.
Definition 2: Error – Fault – Failure

Example:

Input: \( x = [2, 7, 0] \)  
Expected (=correct) result?
Actual result?

```java
public static int numZero (int[] x) {
    // Effects: if x==null throw NullPointerException
    //          else return the number of occurrences of 0 in x
    int count = 0;
    for (int i = 1; i < x.length; i++) {
        if (x[i] == 0) {
            count++;
        }
    }
    return count;
}
```

Input: \( x = [2, 7, 0] \)  
Expected (=correct) result?
Actual result?
Definition 2: Error – Fault – Failure

Example:

Input: \( x = [2, 7, 0] \)
Expected (=correct) result: 1
Actual result: 1

```
public static int numZero (int[] x) {
    // Effects: if x==null throw NullPointerException
    //           else return the number of occurrences of 0 in x
    int count = 0;
    for (int i = 1; i < x.length; i++) {
        if (x[i] == 0) {
            count++;
        }
    }
    return count;
}
```

Failure? Error? Fault?
Definition 2: Error – Fault – Failure

Example:

Input: x = [2, 7, 0]
Expected (=correct) result: 1
Actual result: 1

Failure=no Error? Fault?

```java
public static int numZero (int[] x) {
   // Effects: if x==null throw NullPointerException
   //          else return the number of occurrences of 0 in x
   int count = 0;
   for (int i = 1; i < x.length; i++) {
      if (x[i] == 0) {
         count++;
      }
   }
   return count;
}
```
Definition 2: Error – Fault – Failure

Example:

Input: \( x = [2, 7, 0] \)   Error?

<table>
<thead>
<tr>
<th>State 1:</th>
<th>State 2:</th>
<th>State 3:</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC=public static ...</td>
<td>PC=int count = ...</td>
<td>PC=for (int i = ...</td>
</tr>
<tr>
<td>x=[2, 7, 0]</td>
<td>x=[2, 7, 0]</td>
<td>x=[2, 7, 0]</td>
</tr>
<tr>
<td>count=?</td>
<td>count=0</td>
<td>count=0</td>
</tr>
<tr>
<td>i=?</td>
<td>i=1</td>
<td>i=1</td>
</tr>
</tbody>
</table>

public static int numZero (int[] x) {
// Effects: if x==null throw NullPointerException
//          else return the number of occurrences of 0 in x
    int count = 0;
    for (int i = 1; i < x.length; i++) {
        if (x[i] == 0) {
            count++;
        }
    }
    return count;
}
Definition 2: Error – Fault – Failure

Example:

State 1:
PC=public static …
x=[2, 7, 0]
count=?
i=?

Input: $x = [2, 7, 0]$  Error=yes Fault?

State 2:
PC=int count = …
x=[2, 7, 0]
count=0
i=?

State 3:
PC=for (int i = …
x=[2, 7, 0]
count=0
i=1

---

```
public static int numZero (int[] x) {
    // Effects: if x==null throw NullPointerException
    //          else return the number of occurrences of 0 in x
    int count = 0;
    for (int i = 1; i < x.length; i++) {
        if (x[i] == 0) {
            count++;
        }
    }
    return count;
}
```

Program state: $x$, $i$, $count$, $PC$
Definition 2: Error – Fault – Failure

Example:

Input: $x = [2, 7, 0]$

<table>
<thead>
<tr>
<th>State 1:</th>
<th>State 2:</th>
<th>State 3:</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC=public static …</td>
<td>PC=int count = …</td>
<td>PC=for (int i = …</td>
</tr>
<tr>
<td>x=[2, 7, 0]</td>
<td>x=[2, 7, 0]</td>
<td>x=[2, 7, 0]</td>
</tr>
<tr>
<td>count=?</td>
<td>count=0</td>
<td>count=0</td>
</tr>
<tr>
<td>i=?</td>
<td>i=1</td>
<td>i=1</td>
</tr>
</tbody>
</table>

Program state: $x, i, count, PC$

public static int numZero (int[] $x$) {
    // Effects: if $x$==null throw NullPointerException
    //          else return the number of occurrences of 0 in $x$
    int count = 0;
    for (int $i = 1; i < x.length; i++) {
        if (x[$i] == 0) {
            count++;
        }
    }
    return count;
}
Definition 2: Error – Fault – Failure

Example:

Input: \( x = \{2, 7, 0\} \)

Expected (=correct) result: 1

Actual result: 1

Failure=no Error=yes Fault=yes

Program state: \( x, i, count, PC \)

```
public static int numZero (int[] x) {
    // Effects: if x==null throw NullPointerException
    //          else return the number of occurrences of 0 in x
    int count = 0;
    for (int i = 1; i < x.length; i++) {
        if (x[i] == 0) {
            count++;
        }
    }
    return count;
}
```
Definition 2: Error – Fault – Failure

Example:

Input: \( x = [0, 7, 2] \)
Expected (=correct) result: ?
Actual result: ?
Failure=? Error=? Fault=?

Program state: \( x, i, \text{count}, \text{PC} \)

```java
public static int numZero (int[] x) {
    // Effects: if \( x == \text{null} \) throw NullPointerException
    // else return the number of occurrences of 0 in \( x \)
    int count = 0;
    for (int i = 1; i < x.length; i++) {
        if (x[i] == 0) {
            count++;
        }
    }
    return count;
}
```
Definition 2: Error – Fault – Failure

Example:

Input: \( x = [0, 7, 2] \)

Expected (=correct) result: 1

Actual result: 0

Failure=yes Error=yes Fault=yes

Program state: \( x, i, count, PC \)

```java
public static int numZero (int[] x) {
    // Effects: if x==null throw NullPointerException
    //          else return the number of occurrences of 0 in x
    int count = 0;
    for (int i = 1; i < x.length; i++) {
        if (x[i] == 0) {
            count++;
        }
    }
    return count;
}
```
Types of Defects (Faults)

Requirements defects
Design defects
Coding defects
Testing defects

Orthogonal Defect Classification (ODC)

Fig 3.2
Test Case

• A test case is a set of conditions or variables under which a tester will determine whether a system under test satisfies requirements or works correctly.

• Templates and examples of formal test case documentation can be found here:

  http://softwaretestingfundamentals.com/test-case/
**Test Case**

A **Test Case** consists of:
- A set of inputs + expected outputs
- Execution conditions
  
  Example of ‘execution condition’:
  When pressing the ‘save’ button of a word processor, what happens depends on what you did previously (e.g., what you typed in or deleted)

---

**Test Suite** = set of Test Cases

**Test Data** = input to a Test Case

**Test Oracle** = condition that determines whether a test case passed or failed (⇒ fail happens if actual output is different from expected output)

**Test Verdict** = decision of whether a test passed or failed
Test Case – Examples

- **A test case** consists of:
  - A set of inputs
  - Execution conditions
  - Expected outputs

<table>
<thead>
<tr>
<th>TC ID</th>
<th>TC Name</th>
<th>Description</th>
<th>Steps</th>
<th>Expected result</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC1</td>
<td>TC1_FileUpdate Scenario1</td>
<td>Validate that system updates the file when first character is A and second character is a digit.</td>
<td>1. Open the application. 2. Enter first character as &quot;A&quot; 3. Enter second character as a digit</td>
<td>File is updated.</td>
</tr>
<tr>
<td>TC2</td>
<td>TC2_FileUpdate Scenario2</td>
<td>Validate that system updates the file when first character is B and second character is a digit.</td>
<td>1. Open the application. 2. Enter first character as &quot;B&quot; 3. Enter second character as a digit</td>
<td>File is updated.</td>
</tr>
</tbody>
</table>
Test Oracle

= a mechanism used for determining whether a test has passed or failed

- It provides the expected result for a test, e.g. from
  - Specification document, mathematical formula, program model, other program, historical data, belief of tester, ...
- **Test Verdict** = the actual judgment after a test case terminates – pass/fail/warning/don’t know

Sometimes it is difficult/impossible to find an oracle

- Non-functional quality aspects / Scientific computing
Test Script

• A **Test Script** is a set of instructions (written using a scripting/programming language) that is performed on a system under test to verify that the system performs as expected. Test scripts are used in automated testing.

• Examples of Test Frameworks supporting test scripting:
  • JUnit, Selenium, Sikuli, …
Test Script – Examples

JUnit

@Test
public void shortRegularRental() {
    Customer customer = new Customer("Cust");
    Movie movie = new Movie("Groundhog Day", REGULAR);
    Rental rental = new Rental(movie, 2); // 2 days rental = short
    customer.addRental(rental);

    String expected = "Rental Record for Cust\n";
    expected += "\tGroundhog Day\t2.0\n";
    expected += "Amount owed is 2.0\n";
    expected += "You earned 1 frequent renter points";

    Assert.assertEquals(expected, customer.statement());
}

Sikuli

def sample_test_script (self):
    type ("TextA")
    click (ImageButtonA)
    assertExist (ImageResultA)

@Test
public void shortRegularRental() {
    Customer customer = new Customer("Cust");
    Movie movie = new Movie("Groundhog Day", REGULAR);
    Rental rental = new Rental(movie, 2); // 2 days rental = short
    customer.addRental(rental);

    String expected = "Rental Record for Cust\n";
    expected += "\tGroundhog Day\t2.0\n";
    expected += "Amount owed is 2.0\n";
    expected += "You earned 1 frequent renter points";

    Assert.assertEquals(expected, customer.statement());
}
Why do faults occur in software? (Ch 3)

• Software is written by humans
  – Who know something, but not everything
  – Who have skills, but aren’t perfect
  – Who don’t usually use rigorous methods
  – Who do make mistakes (errors)

• Under increasing pressure to deliver to strict deadlines
  – No time to check, assumptions may be wrong
  – Systems may be incomplete

• Software is complex, abstract and invisible
  – Hard to understand
  – Hard to see if it is complete or working correctly
  – No one person can fully understand large systems
  – Numerous external interfaces and dependencies
Principle 1 – Purpose of testing

• Testing is the process of exercising a software component using a selected set of test cases, with the intent of
  – Revealing defects
  – Evaluating quality

• “The testing paradox”

  Testing can’t show that defects don’t exist
  --
  The more you test software, the more immune it becomes to your tests
Principles (cont.)

2: **A good test case** – When the test objective is to detect defects, then a good test case is one that has high probability of revealing a yet undetected defect(s)

3: **Test result** – The results should be inspected meticulously

4: **Expected output** – A test case must contain the expected output
Principles (cont.)

5: **Input** – Test cases should be developed for both valid and invalid input conditions

6: **Fault content estimation** – The probability of the existence of additional defects in a software component is proportional to the number of defects already detected in that component

7: **Test organization** – Testing should be carried out by a group that is independent of the development group
Principles (cont.)

8: **Repeatable** – Tests must be repeatable and reusable

9: **Planned** – Testing should be planned

10: **Life cycle** – Testing activities should be integrated into the software life cycle

11: **Creative** – Testing is a creative and challenging task
Recommended Textbook Exercises

- Chapter 1
  - 2, 3
- Chapter 2
  - 1, 3, 5
- Chapter 3
  - 1, 2, 3, 6
Structure of Lecture 1

• Introduction and Motivation
• Course Information
• Definitions and Principles
• Lab 1
Lab 1 – Debugging

System 1

Issue 1
Issue 2
Issue 3

Faults?
Faults?
Faults?

System 2

Issue 1
Issue 2

Faults?
Faults?

Localization steps & heuristics for each fault

Use IntelliJ Debugger

NEW!
### Issue Report

- **Example Report**
  - Admin Data
  - Short Description
  - Reproduction Steps (input)
  - Expected vs Actual Result
  - Additional Information (screen shots, stack traces, etc.)
  - Comments / Discussion
Next Week

• Lab 1:
  – Debugging

• Lecture 2:
  – Black-Box Testing Techniques

• In addition to do:
  – Read textbook chapters 1-3 (available via OIS)