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

Spring 2017
Structure of Lecture 1

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

(\textit{Dynamic}) Testing:
- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing

Reviews:
- Requirement Review
- Design Review
- Code Review
- Deployment Plan Review
- Test Plan Review
- Test Cases Review
What is Software Testing?
(Static & Dynamic)

Confirm quality (pass-test) vs.
Find defects (fail-test)

(Testing paradox)
What is Software Testing?

The process of evaluating a program or a system

Verifying

Validating
What is Software Testing?

The process of evaluating a program or a system

1-Verifying

2-Validating

V... ... that the right product has been developed

Answer choices: 1 or 2

Poll link: http://poll.fm/5r279
Verification or Validation?

V... that the right product has been developed

Validating that the right product has been developed

Correct

Verifying that the right product has been developed

<table>
<thead>
<tr>
<th></th>
<th>COUNT</th>
<th>PERCENT COUNTRY</th>
<th>PERCENT OVERALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Verifying</td>
<td>9</td>
<td>64.29%</td>
<td>64.29%</td>
</tr>
<tr>
<td>2: Validating</td>
<td>5</td>
<td>35.71%</td>
<td>35.71%</td>
</tr>
</tbody>
</table>
What is Software Testing?

The process of evaluating a program or a system

**Verifying** that the product has been developed right

**Validating** that the right product has been developed
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?

Evaluation Items:
- Requirement, actual product/software

Activities:
- Requirements review, acceptance testing
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.

http://SoftwareTestingFundamentals.com/
STLC integrated with SDLC

Actual Needs and Constraints

User Acceptance (alpha, beta test)

Delivered Package

System Specifications

System Test

Analysis / Review

Subsystem Design/Specs

Integration Test

Analysis / Review

Subsystem

Unit/Component Specs

Module Test

User review of external behavior as it is determined or becomes visible

(c) 2008 Mauro Pezzè & Michal Young
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

The testing cycle V model:
- Business case
- Requirement
- System specifications
- System design
- Components design
- Build components
- Unit testing
- Integration testing
- System testing
- User acceptance
- Release 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?

Correct
Test Complexity

Example:
- 30 variables, 2 levels
  \[ 2^{30} \approx 10^9 \]
  combinations to test
- 5 tests/second ->
  214748364.8 sec or
  **6.8 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), 2nd year
• Credits: 3 ECTS
• Prerequisite:
  – Compulsory: MTAT.03.094 Software Engineering (6 ECTS, 4 CP)
  – Recommended: MTAT.03.130 Object-oriented Programming (6 ECTS, 4 CP)
• Work load:
  – Lectures (incl. practical work): 32 hours – incl. labs and exam
  – Independent work: 46 hours
• Assessment:
  – 6 Lab Assignment (work in pairs) – 60% of grade (~8 h per lab)
  – Exam (written) – 40% of grade (~30 h)
• 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 and introduces various testing strategies and types of testing.

• It will also give an overview of different software defects, software defect management, and organizational aspects of software testing.
Goals: Knowledge & Understanding

For a passing grade the student must

- be able to **define** basic **concepts and principles** within software testing
- **give an account** of the most common **techniques** for software testing
- be able to describe the **relation** between the **software testing process** and other processes in the product lifecycle
Goals: Skills and Abilities

For a passing grade the student must

• be able to **apply a systematic approach** for software testing
  – be able to **plan the testing** (incl. design of **test cases**) of a small software system
  – be able to **test a software system** based on a chosen test strategy
Goals: Judgement and Approach

For a passing grade the student must

• understand the **complexity** of the task to test a software system and

• be aware of the **costs** and the impact of different **testing activities** during the development of a software product.
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 (09.02) - Introduction to Software Testing
- Lecture 2 (16.02) - Black-Box Testing Techniques
- Lecture 3 (23.02) - White-Box Testing Techniques
- Lecture 4 (02.03) - Static Analysis & Defect Estimation
- Lecture 5 (09.03) - Lifecycle, Levels, Tools
- Lecture 6 (16.03) - Industry Guest Lecture
- Lecture 7 (04.05) – Documentation, Organisation, Exam preparation

Overview

Technical

Technical / Managerial

Managerial
## Lectures & Labs Schedule

| W24 | W25 | W26 | W27 | W28 | W29 | W30 | W31 | W32 | W33 | W34 | W35 | W36 | ...
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Lec 7</td>
<td></td>
</tr>
<tr>
<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Submit HW 1 → Submit HW 2 → Submit HW 3 → Submit HW 4 → Submit HW 5 → Submit HW 6
Lab Sessions (J. Liivi 2-... various rooms)

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

1. Issue reporting (10%)
2. Black-box & White-box testing (10%)
3. Mutation testing (10%)
4. Document inspection and defect estimation (10%)
   - Bonus task (2.5%)
5. Static code analysis (10%)
6. Automated GUI Testing (Regression Testing) (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 (40%)
• Based on textbook, lectures and lab sessions
• Open book & open laptop

Dates:
• Exam 1: ??-May-2015 14:15-16:15 (Liivi 2-????; limit: 40 stud.)
• Exam 2: ??-May-2015 14:15-16:15 (Liivi 2-????; limit: 40 stud.)
• Exam 3: ??-May-2015 14:15-16:15 (Liivi 2-????; limit: 40 stud.)
+ re-take exam
Literature

– Textbook:
  • Ilene Burnstein: Practical Software Testing

– Additional reading materials posted on course web
Software Testing

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

TA: Kristiina Rahkema (rahkema at ut dot ee)
TA: Margus Luik (mar6luik at gmail dot com)
TA: Faiz Ali Shah (faizallshah at gmail dot com)

Lectures (begin in week 24 of the academic year, on 09-Feb-2017):
- 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, 35, (37) - Kristiina
- Group 2: Tuesday 14.15 - 16.00, r207 - week 25, 27, 29, 31, 33, 35, (37) - Margus
- Group 3: Wednesday 14.15 - 16.00, r202 - week 25, 27, 29, 31, 33, 35, (37) - Margus
- Group 4: Wednesday 14.15 - 16.00, r207 - week 25, 27, 29, 31, 33, 35, (37) - Kristiina
- Group 5: Tuesday 14.15 - 16.00, r404 - week 25, 27, 29, 31, 33, 35, (37) - Faiz

Exams:
- Exam 1: XXXday, ??-May-2015, 14:15-16:30 (J. Liivi 2-???) - capacity limit: 35
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

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: \( x = [2, 7, 0] \)
Definition 2: Error – Fault – Failure

Example:

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

Failure? 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]

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;
}
```

Program state: \( x, i, count, PC \)
Definition 2: Error – Fault – Failure

Example:

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

Input: \( x = [2, 7, 0] \)  \( \text{Error} = \text{yes} \)  \( \text{Fault?} \)

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

\begin{verbatim}
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;
}
\end{verbatim}
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=yes

Program state: x, i, count, PC

State 2:
PC=int count = ...
x=[2, 7, 0]
count=0
i=1

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;
}
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

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

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

```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)
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 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

```python
@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 += "  Groundhog Day	2.0\n";
    expected += "  Amount owed is 2.0\n";
    expected += "  You earned 1 frequent renter points"
    Assert.assertEquals(expected, customer.statement());
}
```

Sikuli

```python
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 += "  Groundhog Day	2.0\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 – Issue (Defect) Reporting

• Issues / Defects are a valuable asset – if reported right
• Analysis of defect reports facilitates:
  – Decision support:
    • Assignment, time/effort allocation, prioritization, etc.
  – Prediction:
    • Severity, Resolution time/effort, etc.
  – Defect prevention:
    • Defect classification → typical / recurring defects
Lab 1 – Issue (Defect) Reporting

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

- Tool: TargetProcess
Lab 1 – Issue (Defect) Reporting

• Workflow

• Regression Test
  – ATM V1 → V2
Next Week

• Lab 1:
  – Issue Reporting

• Lecture 2:
  – Black-Box Testing Techniques

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