Lecture 9:
Verification & Validation (Testing) I

Fall 2021

Dietmar Pfahl
email: dietmar.pfahl@ut.ee
Schedule of Lectures

Week 01: Introduction to SE
Week 02: Requirements Engineering I
Week 03: Requirements Engineering II
Week 04: Analysis
Week 05: Development Infrastructure
Week 06: Continuous Development and Integration
Week 07: Project Estimation / Architecture and Design I
Week 08: Architecture and Design II

Week 09: Verification and Validation I
Week 10: Verification and Validation II
Week 11: Refactoring (and TDD)
Week 12: Agile/Lean Methods
Week 13: Industry Guest Lecture
Week 14: Course wrap-up, review and exam preparation
Week 15: Reserve time slot (no lecture scheduled as of today)
Structure of Lecture 9

• Testing Basics
• Testing Levels
• Testing Methods
• Testing Types
• Testing Artefacts
• Metrics
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 Quality – Dimensions (1)

- **Accessibility**: The degree to which software can be used comfortably by a wide variety of people, including those who require assistive technologies like screen magnifiers or voice recognition.
- **Compatibility**: The suitability of software for use in different environments like different Operating Systems, Browsers, etc.
- **Concurrency**: The ability of software to service multiple requests to the same resources at the same time.
- **Efficiency**: The ability of software to perform well or achieve a result without wasted energy, resources, effort, time or money.
- **Functionality**: The ability of software to carry out the functions as specified or desired.
- **Installability**: The ability of software to be installed in a specified environment.
- **Localizability**: The ability of software to be used in different languages, time zones etc.
Software Quality – Dimensions (2)

- **Maintainability**: The ease with which software can be modified (adding features, enhancing features, fixing bugs, etc).
- **Performance**: The speed at which software performs under a particular load.
- **Portability**: The ability of software to be transferred easily from one location to another.
- **Reliability**: The ability of software to perform a required function under stated conditions for stated period of time without any errors.
- **Scalability**: The measure of software’s ability to increase or decrease in performance in response to changes in software’s processing demands.
- **Security**: The extent of protection of software against unauthorized access, invasion of privacy, theft, loss of data, etc.
- **Testability**: The ability of software to be easily tested.
- **Usability**: The degree of software’s ease of use.
Software Quality – Dimensions (2)

- **Maintainability**: The ease with which software can be modified (adding features, enhancing features, fixing bugs, etc).
- **Performance**: The speed at which software performs under a particular load.
- **Portability**: The ability of software to be transferred easily from one location to another.
- **Reliability**: The ability of software to perform a required function under stated conditions for stated period of time without any errors.
- **Scalability**: The measure of software’s ability to increase or decrease in performance in response to changes in software’s processing demands.
- **Security**: The extent of protection of software against unauthorized access, invasion of privacy, theft, loss of data, etc.
- **Testability**: The ability of software to be easily tested.
- **Usability**: The degree of software’s ease of use.

Safety?
Software Product Quality Model
– ISO 25010 Standard

Safety?
Software Product Quality Model
– ISO 25010 Standard
Software Quality Assurance (SQA)

versus

Software Quality Control (SQC)
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 (static/dynamic):
- Reviews (manual/automated)
- Testing (manual/automated)

**Reviews:**
- Requirement Review
- Design Review
- Code Review
- Deployment Plan Review
- Test Plan Review
- Test Cases Review

**Testing:**
- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing
Verification versus Validation
Validation versus Verification

- Requirements Backlog (e.g., User Stories)
- Work Product
- Work Product
- Work Product
- Development Process
- End Product (i.e., the software that is delivered/deployed)
Validation versus Verification

Requirements Backlog
(e.g., User Stories)

Work Product
Work Product
...
Work Product

Development Process

End Product
(i.e., the Software that is delivered/deployed)
Validation versus **Verification**

- **Requirements Backlog** (e.g., User Stories)
- **Work Product**
- **Work Product**
- **Work Product**...
- **End Product** (i.e., the Software that is delivered/deployed)

**Development Process**
Verification

Definition
• The process of evaluating work-products (not the final end 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?
Software Development Life Cycle (SDLC)

SDLC in summary:
- Project Planning
- Requirements Development
- Estimation
- Scheduling
- Design
- Coding
- Test Build/Deployment
- Unit Testing
- Integration Testing
- User Documentation
- System Testing
- Acceptance Testing
- Production Build/Deployment
- Release
- Maintenance
Software Development Life Cycle (SDLC)

- The SDLC, or Software Development Process, defines the steps/stages/phases in the building of software.

- Various kinds of software development models exist, e.g.:
  - Waterfall model
  - Spiral model
  - Iterative and incremental development (like ‘Unified Process’ and ‘Rational Unified Process’)
  - Agile development (like ‘Extreme Programming’ and ‘Scrum’)

**SDLC in summary:**
- Project Planning
- Requirements Development
- Estimation
- Scheduling
- Design
- Coding
- Test Build/Deployment
- Unit Testing
- Integration Testing
- User Documentation
- System Testing
- Acceptance Testing
- Production Build/Deployment
- Release
- Maintenance
Software Testing Life Cycle (STLC)

- The STLC defines the steps/stages/phases in testing of software.
# Software Testing Life Cycle (STLC)

<table>
<thead>
<tr>
<th>Phase</th>
<th>Activity</th>
<th>Deliverables</th>
<th>Attitude needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements/ Design</td>
<td>You review software requirements/design (if existing)</td>
<td>Review Reports (listing the defects)</td>
<td>Curiosity</td>
</tr>
<tr>
<td>Review</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test Planning</td>
<td>After gathering a general idea of what needs to be tested, you ‘plan’ for the tests</td>
<td>Test Plan, Test Estimation, Test Schedule</td>
<td>Farsightedness</td>
</tr>
<tr>
<td>Test Designing</td>
<td>You design/detail your tests on the basis of detailed requirements/design of the software</td>
<td>Test Cases, Test Data, Test Scripts, Requirements, Traceability Matrix</td>
<td>Creativity</td>
</tr>
<tr>
<td>Test Environment Setup</td>
<td>You setup the test environment (tools)</td>
<td>Test Environment</td>
<td>Interest in test technology</td>
</tr>
<tr>
<td>Test Execution</td>
<td>You execute your Test Cases/Scripts in the Test Environment to see whether they pass</td>
<td>Test Results (intermediate), Defect Reports</td>
<td>Patience</td>
</tr>
<tr>
<td>Test Reporting</td>
<td>You prepare various reports for various stakeholders</td>
<td>Test Results (final), Test/Defect Metrics, Test Closure Report</td>
<td>Accuracy, Diplomacy</td>
</tr>
</tbody>
</table>
STLC integrated with SDLC

Actual Needs and Constraints

User Acceptance (alpha, beta test)

Delivered Package

System Test

Analysis / Review

System Specifications

Subsystem

Subsystem Design/Specs

Integration Test

Analysis / Review

Unit/Component Specs

Integration Test

Module Test

Unit/Components

Test Levels

User review of external behavior as it is determined or becomes visible
More vocabulary …
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 passes or fails
  (-> ‘fail’ happens if actual output is different from expected output)
Test Verdict = decision of whether a test has passed or failed
Test Case – Recommendations

• As far as possible, write test cases in such a way that you test only one thing at a time. Do not overlap or complicate test cases. Try to make your test cases ‘atomic’.

• Ensure that all **positive scenarios** and **negative scenarios** are covered.

• **Language:**
  • Write in simple and easy to understand language.
  • Use active voice: Do this, do that.
  • Use exact and consistent names (of forms, fields, etc).

• **Characteristics of a good test case:**
  • **Accurate:** Exacts the purpose.
  • **Economical:** No unnecessary steps or words.
  • **Traceable:** Capable of being traced to requirements.
  • **Repeatable:** Can be used to perform the test over and over.
  • **Reusable:** Can be reused if necessary.
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);
    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)

Oracle
Input
Expected Output
Oracle
Actual Output
What is a ‘Bug’ in SE?
First ‘Computer Bug’ in 1947

The term "bug" was used in an account by computer pioneer Grace Hopper, who publicized the cause of a malfunction in an early electromechanical computer. (Harvard’s Mark II relay computer)

Source: https://en.wikipedia.org/wiki/Software_bug
What is a Bug in SE?

Twins
What is a Bug in SE?

if amountOf(baby) > 1
    answer = "Twins";
    print(answer);
...
What is a Bug in SE?

if amountOf(baby) > 1
   answer = "Twins";
   print(answer);
...
What is a Bug in SE?

if amountOf(baby) > 1
    answer = "Baybies";
if equals(baby[0], baby[1])
    answer = "Twins";
print(answer);
...

Fault?
What is a Bug in SE?

Error?

Failure?

Fault?

```java
... if amountOf(baby) > 1
    answer = "Twins";
    if equals(baby[0], baby[1])
        answer = "Twins";
    print(answer);
...```
Definition 1: Error – Fault – Failure (according to IEEE Standard)

- **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.
Definition 1: Error – Fault – Failure (according to IEEE Standard)

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

Source: Fig 3.1 in I. Burnstein: Practical Software Testing
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:

```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;
}
```

Inputs: Correct (=Expected) result?
- x = [2,7,0] Actual result?
- x = [0,7,2] Fault? Error? Failure?

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

Example:

```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;
}
```

Inputs:

- Correct (=Expected) result? ?
- x = [2,7,0] Actual result? ?

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

Example:

```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;
}
```

Inputs:
Correct (=Expected) result? 1
x = [2,7,0] Actual result? ?
Definition 2: Error – Fault – Failure

Example:

```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;
}
```

Inputs:

<table>
<thead>
<tr>
<th>Correct (=Expected) result?</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = [2,7,0]</td>
<td>1</td>
</tr>
<tr>
<td>Actual result?</td>
<td>1</td>
</tr>
</tbody>
</table>

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

```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;
}
```

Inputs: Correct (=Expected) result?

- `x = [2,7,0]` Actual result?
- `x = [0,7,2]` Fault? Error? Failure?

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

Example:

```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;
}
```

Inputs: Correct (=Expected) result? 1
x = [2,7,0] Actual result? 1
Fault? Error? Failure? Yes / ? / No

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

Example:

```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;
}
```

Inputs:

- Correct (=Expected) result? 1
- Actual result? 1
- Fault? Error? Failure? Yes / Yes / No

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

Example:

```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;
}
```

Inputs:

<table>
<thead>
<tr>
<th>Correct (=Expected) result?</th>
<th>?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual result?</td>
<td>?</td>
</tr>
</tbody>
</table>

x = [0,7,2]


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

Example:

```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;
}
```

Inputs:
Correct (=Expected) result? 1
Actual result? 0
x = [0,7,2]

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

Example:

```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;
}
```

Inputs:
- Correct (=Expected) result? 1
- Actual result? 0
- x = [0,7,2]
- Fault? Error? Failure? Yes / Yes / Yes

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

We have seen ...

Fault = Yes
Error = Yes
------------------
Failure = No
or
Failure = Yes
Definition 2: Error – Fault – Failure

Could any of this happen?

Fault = No  
Error = No  
---------------
Failure = No  
or  
Failure = Yes

Fault = No  
Error = Yes  
---------------
Failure = No  
or  
Failure = Yes

Fault = Yes  
Error = No  
---------------
Failure = No  
or  
Failure = Yes
Definition 2: Error – Fault – Failure

Could any of this happen?

Fault = No  
Error = No
-----------------
Failure = No  
or  
Failure = Yes

Fault = No  
Error = Yes
-----------------
Failure = No  
or  
Failure = Yes

Fault = Yes  
Error = No
-----------------
Failure = No  
or  
Failure = Yes

?
Definition 2: Error – Fault – Failure

Example:

```java
public static int numZero (int[] x) {
    // Effects: if x==null throw NullPointerException
    //          else return the index of the 1st occurrence of 0 in x

    for (int i = 0; i < x.length-1; i++) {
        if (x[i] == 0) {
            return i;
        }
    }
    return -1;
}
```

Inputs: Correct (=Expected) result? ?
         Actual result? ?

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

Example:

```java
public static int numZero (int[] x) {
    // Effects: if x==null throw NullPointerException
    //          else return the index of the 1st occurrence of 0 in x

    for (int i = 0; i < x.length-1; i++) {
        if (x[i] == 0) {
            return i;
        }
    }
    return -1;
}
```

Inputs:  
Correct (=Expected) result? 0
Actual result? 0

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

Example:

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

Inputs: Correct (=Expected) result? 0
        Actual result?
        Fault? Error? Failure? Yes / ? / No

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

Example:

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

Inputs: Correct (=Expected) result? 0
         Actual result? 0
         x = [0,7,2]  Fault? Error? Failure? Yes / ? / No

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

Summary of the four possible situations:

<table>
<thead>
<tr>
<th>Fault</th>
<th>Error</th>
<th>Failure</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Failure = Yes
or
Failure = No
Definition 2: Error – Fault – Failure

Summary of the four possible situations:

Fault = No  Error = No  FAILURE = No

Fault = Yes  Error = No  FAILURE = No

Fault = Yes  Error = Yes  FAILURE = Yes

or  FAILURE = No  or  FAILURE = No
Definition 2: Error – Fault – Failure

Summary of the four possible situations:

- Fault = No, Error = No, Failure = No
- Fault = Yes, Error = No, Failure = No
- Fault = Yes, Error = Yes, Failure = No
- Fault = Yes, Error = Yes, Failure = Yes

‘No failure’ when testing does not imply ‘no fault/error’!
Other often used terms for *failure* …

**Incident**
- Often used when something suspicious has happened but it is not yet clear what it is
- It is a symptom that something is wrong
- Alerts the tester or user that a failure might come

**Issue**
- Often used in a broad sense stating that something is going on but without making claims about where it comes from, if it is a failure due to some fault, or whether it should be fixed.
- Sometimes even referring to tasks or to requirements (e.g., in issue trackers)
Flaky Test

A test that sometime passes and sometimes fails due to environmental conditions that cannot be (fully) controlled during testing

Typical Causes:
• Cached data, order of tests (no proper cleanup), assertion timing inconsistent with global state, interaction with (unstable or buggy) 3rd party system, time bomb (accessing local time inconsistently)

Tips on how to deal with it:
• https://hackernoon.com/flaky-tests-a-war-that-never-ends-9aa32fdef359
Structure of Lecture 9

- Testing Basics
- Testing Levels
- Testing Methods
- Testing Types
- Testing Artefacts
- Metrics

Next week!
Next Lecture

• **Date/Time:**
  - Friday, 05-Nov, 10:15-12:00

• **Topic:**
  - Verification and Validation (Testing) II

• **Labs:**
  - Submit homework assignment 4
  - Start working on homework assignment 5