LTAT.05.003
Software Engineering

Lecture 10:
Verification & Validation (Testing) II

Dietmar Pfahl

email: dietmar.pfahl@ut.ee

Fall 2019

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: 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: *Agile Methods in Industry*

Week 14: Course wrap-up, review and exam preparation

Week 15: Reserve time slot (no lecture scheduled as of today)

Structure of Lecture 10

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

Overview of Testing Types

- Smoke Testing
- Functional Testing
- Usability Testing
- Security Testing
- Performance Testing
- Regression Testing
- Compliance Testing

Security Testing

- Security Testing is a type of software testing that intends to uncover vulnerabilities of the system and determine that its data and resources are protected from possible intruders.

- 4 Focus Areas:
  - **Network security:** This involves looking for vulnerabilities in the network infrastructure (resources and policies).
  - **System software security:** This involves assessing weaknesses in the various software (operating system, database system, and other software) the application depends on.
  - **Client-side application security:** This deals with ensuring that the client (browser or any such tool) cannot be manipulated.
  - **Server-side application security:** This involves making sure that the server code and its technologies are robust enough to fend off any intrusion.

Security Testing

Example of a basic security test:

- Log into the web application.
- Log out of the web application.
- Click the BACK button of the browser, then check if you are asked to log in again or if you are provided the logged-in application.

- Most types of security testing involve complex steps and out-of-the-box thinking but, sometimes, it is simple tests like the one above that help expose the most severe security risks.

```
SELECT * FROM Users WHERE Username='$username' AND Password='$password'
```

```
$username = 1' or '1' = '1
$password = 1' or '1' = '1
SELECT * FROM Users WHERE Username='1' OR '1' = '1' AND
                          Password='1' OR '1' = '1'
```

- The Open Web Application Security Project (OWASP) is a great resource for software security professionals. Be sure to check out the Testing Guide: https://www.owasp.org/index.php/Category:OWASP_Testing_Project

OWASP Top 10 security threats for 2013 were:
- Injection
- Broken Authentication and Session Management
- Cross-Site Scripting (XSS)
- Insecure Direct Object References
- Security Misconfiguration
- Sensitive Data Exposure
- Missing Function Level Access Control
- Cross-Site Request Forgery (CSRF)
- Using Known Vulnerable Components
- Unvalidated Redirects and Forwards

OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	U	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	U	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	☒	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	☒	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

How to avoid SQL injection vulnerability?

Instead of:

```
String query = "SELECT * FROM Users WHERE Username= ”
   + request.getParameter(”username")
   + “AND Password= “
   + request.getParameter(”password");

 try {
   Statement statement = connection.createStatement();
   ResultSet results = statement.executeQuery(query);
 }
```

Which might result in a SQL query string like this:

```
SELECT * FROM Users WHERE Username='1' OR '1' = '1' AND
                                          Password='1' OR '1' = '1’
```

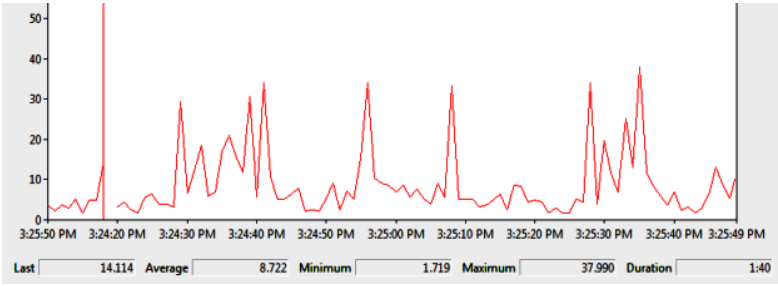
How to avoid SQL injection vulnerability?

Use java ‘prepared statement’:

```
String username = request.getParameter(”username");
String password = request.getParameter(”password");
// perform input validation to detect attacks

String query = "SELECT * FROM Users WHERE Username= ? AND Password= ?";

PreparedStatement pstmt = connection.prepareStatement(query);
pstmt.setString( 1, username);
pstmt.setString( 2, password);

ResultSet results = pstmt.executeQuery( );
```

Example with Hibernate Query Language (HQL) can be found here:
https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

Performance Testing



- Performance Testing is a type of software testing that intends to determine how a system performs in terms of responsiveness and stability under a certain load.

Types:

- **Load Testing** is a type of performance testing conducted to evaluate the behavior of a system at increasing workload.
- **Stress Testing** a type of performance testing conducted to evaluate the behavior of a system at or beyond the limits of its anticipated workload.
- **Endurance Testing** is a type of performance testing conducted to evaluate the behavior of a system when a significant workload is given continuously.
- **Spike Testing** is a type of performance testing conducted to evaluate the behavior of a system when the load is suddenly and substantially increased.

Regression Testing

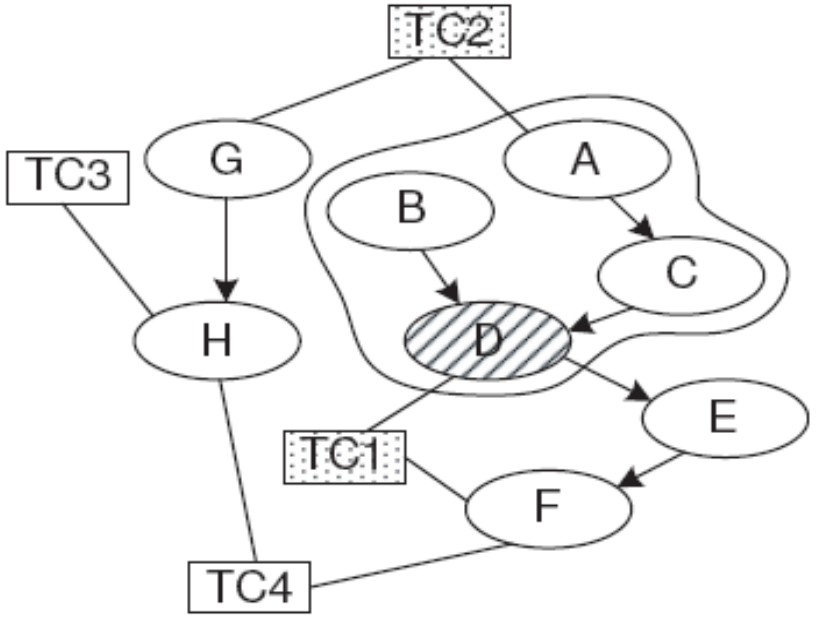
- Regression testing is a type of software testing that intends to ensure that changes (enhancements or defect fixes) to the software have not adversely affected it.

How much?

- In an ideal case, a full regression test is desirable but oftentimes there are time/resource constraints. In such cases, it is essential to do an impact analysis of the changes to identify areas of the software that have the highest probability of being affected by the change and that have the highest impact to users in case of malfunction and focus testing around those areas.

Regression Testing – Retest All

- Assumption:
  - Changes may introduce faults anywhere in the code
- BUT: expensive, prohibitive for large systems
- Reuse existing test suite
- Add new tests as needed
- Remove obsolete tests

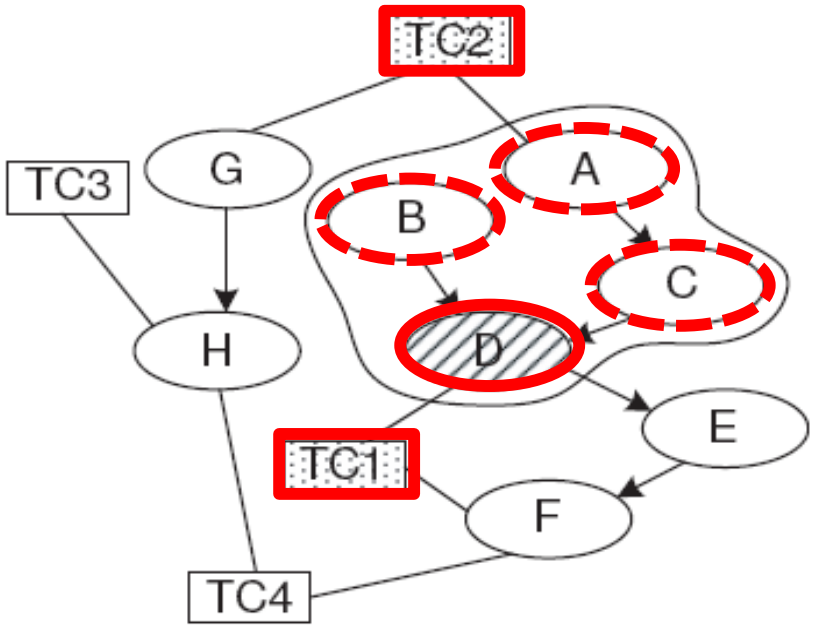


[Skoglund, Runeson, ISESE05]

Regression Testing – Selective Testing

- Conduct impact analysis
  - Only code impacted by change needs to be retested
  - Select tests that exercise such code

- Add new tests if needed
- Remove obsolete tests



Compliance Testing

- Compliance Testing, also known as conformance testing, regulation testing, standards testing, is a type of testing to determine the compliance of a system with internal or external standards.

Checklist:

Lorem ipsum dolor sit amet	✓
consectetur adipiscing elit	✓
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua	✓
Ut enim ad minim veniam	✗
quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat	✓

Structure of Lecture 10

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

Test Documentation

IEEE 829-2008: Standard for Software and System Test Documentation

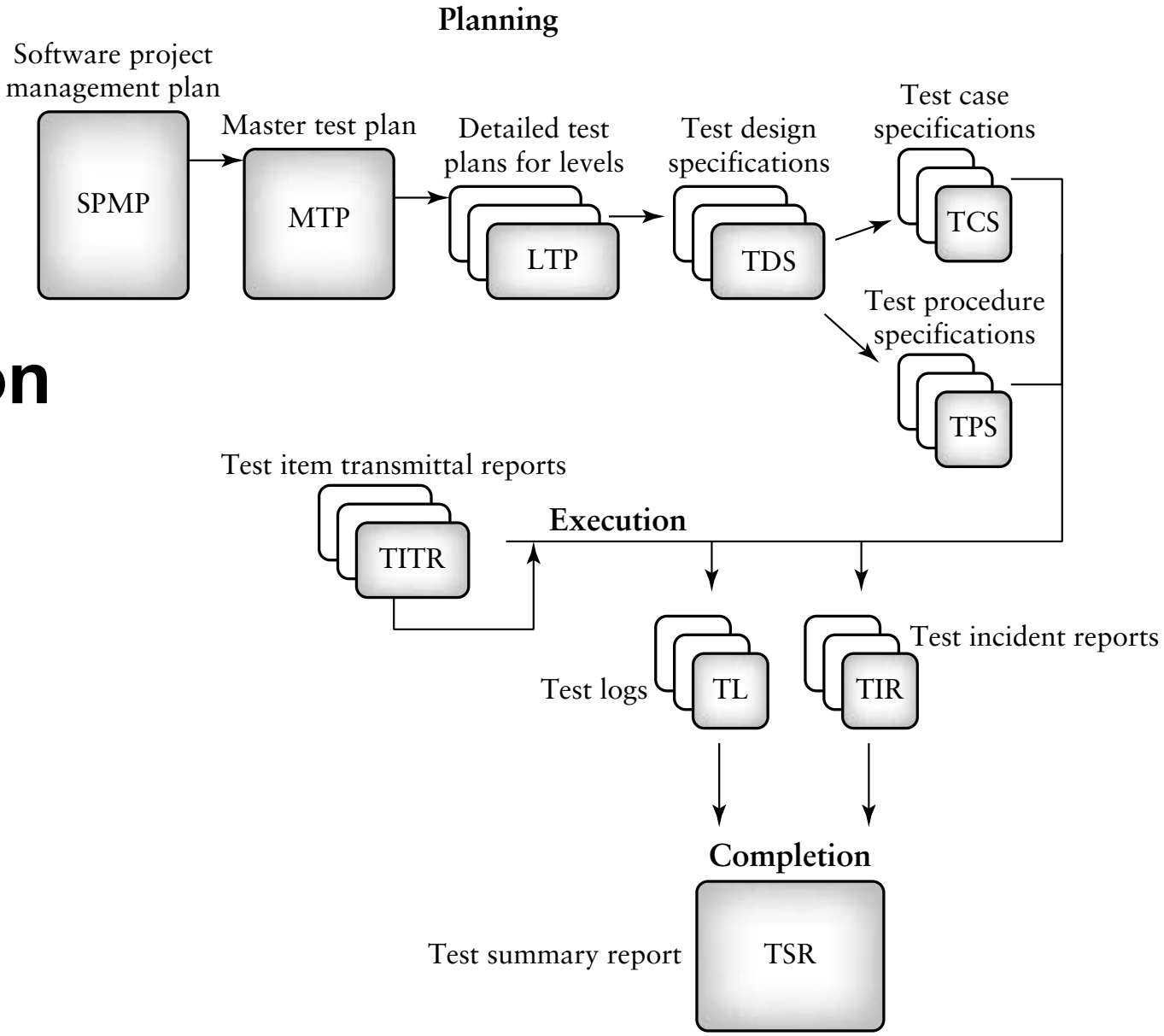
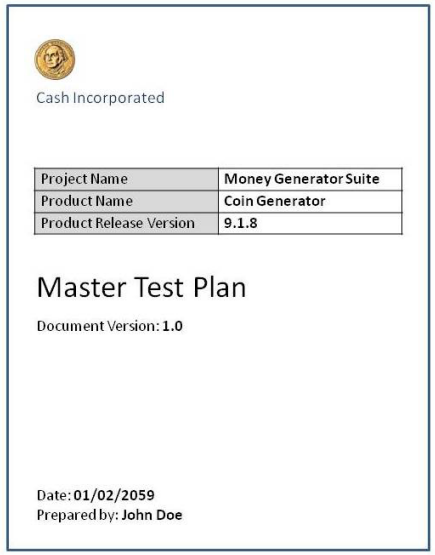


FIG. 7.4

Test Plan

- A Software Test Plan is a document describing the testing scope and activities. It is the basis for formally testing any software/product in a project.


Cash Incorporated

Project Name	Money Generator Suite
Product Name	Coin Generator
Product Release Version	9.1.8

Master Test Plan

Document Version: 1.0

Date: 01/02/2059
Prepared by: John Doe

One can have the following types of test plans:

- **Master Test Plan:** A single high-level test plan for a project/product that unifies all other test plans.
- **Testing Level Specific Test Plans:**
  - Unit Test Plan
  - Integration Test Plan
  - System Test Plan
  - Acceptance Test Plan
- **Testing Type Specific Test Plans:** Plans for major types of testing like Performance Test Plan and Security Test Plan

Test Case – Template

Test Suite ID	The ID of the test suite to which this test case belongs.
Test Case ID	The ID of the test case.
Test Case Summary	The summary / objective of the test case.
Related Requirement	The ID of the requirement this test case relates/traces to.
Prerequisites	Any prerequisites or preconditions that must be fulfilled prior to executing the test.
Test Procedure	Step-by-step procedure to execute the test.
Test Data	The test data, or links to the test data, that are to be used while conducting the test.
Expected Result	The expected result of the test.
Actual Result	The actual result of the test; to be filled after executing the test.
Status	Pass or Fail. Other statuses can be 'Not Executed' if testing is not performed and 'Blocked' if testing is blocked.
Remarks	Any comments on the test case or test execution.
Created By	The name of the author of the test case.
Date of Creation	The date of creation of the test case.
Executed By	The name of the person who executed the test.
Date of Execution	The date of execution of the test.
Test Environment	The environment (Hardware/Software/Network) in which the test was executed.

Test Case – Example

Test Suite ID	TS001
Test Case ID	TC001
Test Case Summary	To verify that clicking the Generate Coin button generates coins.
Related Requirement	RS001
Prerequisites	User is authorized; Coin balance is available.
Test Procedure	Select the coin denomination in the Denomination field. Enter the number of coins in the Quantity field. Click Generate Coin.
Test Data	Denominations: 0.05, 0.10, 0.25, 0.50, 1, 2, 5 Quantities: 0, 1, 5, 10, 20
Expected Result	Coin of the specified denomination should be produced if the specified Quantity is valid (1, 5) A message 'Please enter a valid quantity between 1 and 10' should be displayed if the specified quantity is invalid.
Actual Result	If the specified quantity is valid, the result is as expected. If the specified quantity is invalid, nothing happens; the expected message is not displayed
Status	Fail
Remarks	This is a sample test case.
Created By	John Doe
Date of Creation	01/14/2020
Executed By	Jane Roe
Date of Execution	02/16/2020
Test Environment	OS: Windows Y, Browser: Chrome N

Incident/Issue Report

- In most companies, an issue reporting tool is used and the elements of a report can vary. However, in general, an issue report can consist of the following elements.

ID Unique identifier given to the issue. (Usually Automated)
Project Project name.
Product Product name.
Release Version Release version of the product. (e.g. 1.2.3)
Module Specific module of the product where the issue occurred.
Detected Build Version Build version of the product where the issue was detected (e.g. 1.2.3.5)
Summary Summary of the issue. Keep this clear and concise.
Description Detailed description of the issue. Describe as much as possible but without repeating anything or using complex words. Keep it simple but comprehensive.
Steps to Replicate Step by step description of the way to reproduce the issue. Number the steps.
Actual Result The actual result you received when you followed the steps.
Expected Results The expected results.
Attachments Attach any additional information like screenshots and logs.
Remarks Any additional comments on the issue.
Issue Severity Severity of the issue.
Issue Priority Priority of the issue.
Reported By The name of the person who reported the issue.
Assigned To The name of the person that is assigned to analyze/ fix the issue.
Status The status of the issue.
Fixed Build Version Build version of the product where the issue was fixed (e.g. 1.2.3.9)

Structure of Lecture 10

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

Code Coverage

- The relative amount of covered items with regards to a coverage criterion, e.g., statement, branch, condition, …

  statement_coverage =

  executed_statements / total_number_statements

  branch_coverage =

  executed_branches / total_number_branches

Usage

- To control the comprehensiveness of a test suite
- Often used as test stopping criterion

Defect Density

- Defect Density is the number of confirmed defects detected in software/component during a defined period of development/operation divided by the size of the software/component.

$$defect\_density = number\_confirmed\_defects / size$$

Usage:

- For comparing the relative number of defects in various software components (or software products) so that high-risk components can be identified and resources focused towards them.

Defect Coverage

- Defect Coverage is the number of confirmed defects detected in software/component during a defined period of development/operation divided by the total number of defects.

defect_coverage =
number_confirmed_defects / total_number_defects

Usage:
- To assess the effectiveness of a test suite.
- Might be applied for certain types of defects (e.g., severity=major; priority=high)