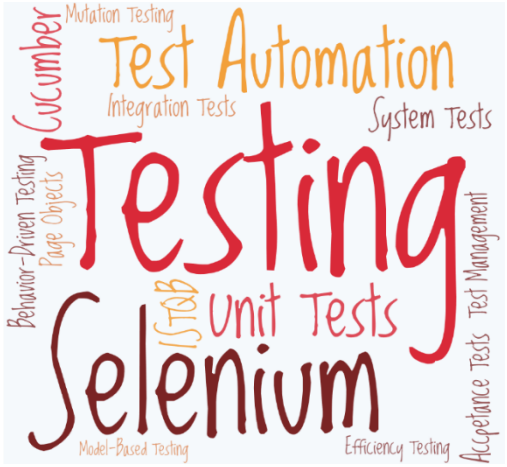
183.290 Software Testing



Testing in the Software Lifecycle

WS 2020

Christina Zoffi

peso@inso.tuwien.ac.at
https://peso.inso.tuwien.ac.at

Content

Testing in the Software Lifecycle

- Testing is **not an isolated task**
- Testing is an **on-going activity** throughout the **software development lifecycle**
- Each **development activity** has a corresponding **testing activity**
- Test design takes place **parallel** to the corresponding development activity
- Testers help **refining requirements** and design and are involved in **reviewing work products**

Testing in the Software Lifecycle

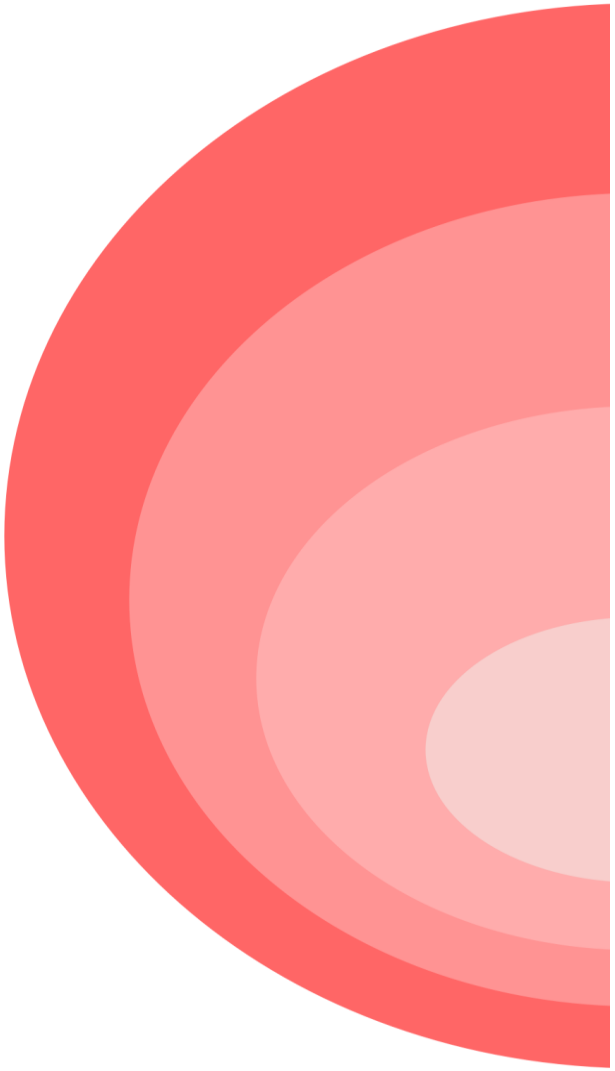
Verification

- Assurance that a component or system **conforms to the specification**
- *Are we building the product right?*

Validation

- Assurance that a system satisfies the **user‘s needs or expectations** and fulfills its **intended use**
- *Are we building the right product?*

Test Levels



Acceptance Testing

System Testing

Integration Testing

Unit Testing
(Component Testing)

Test Levels

- **Test levels** are groups of test activities that are managed and organized together

- Test levels can be applied **sequentially** or **iteratively** depending on the development model

- Test levels need to be tailored to the **project/product environment** and **development lifecycle**

Test Levels – Unit Testing

- **Definition**
  - Testing of individual hardware or software units or groups of related units. (IEEE 610.12-90)

- Also known as **component/module testing**
- Detect defects and verify the functioning of **individual units** (modules, objects or classes) of a system
- May be done in **isolation** from the rest of the system
- Requires access to the code base and development environment (e.g. unit test framework)
- Usually done by developers
- Mostly automated using unit testing frameworks

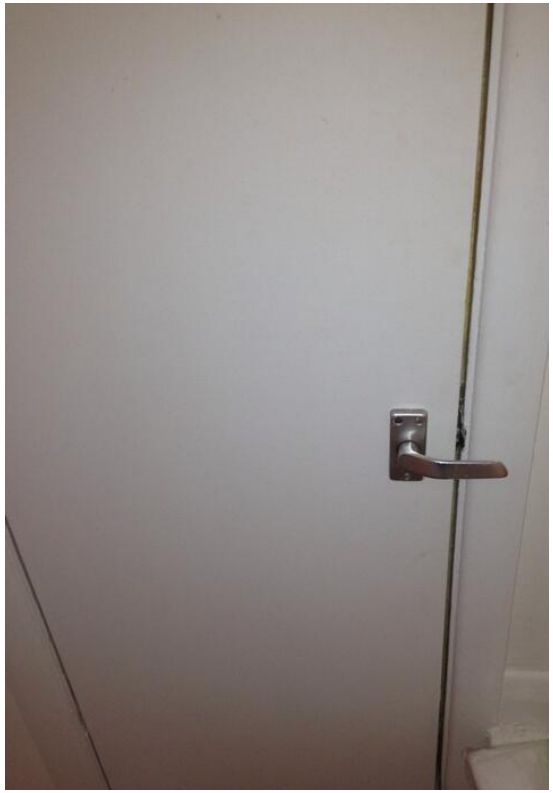
Test Levels – Integration Testing

- **Definition**
  - Testing in which software components, hardware components, or both are combined and tested to evaluate the interaction between them. (IEEE 610.12-90)

- Expose defects in the **interfaces** and **interactions** between **integrated units**
- Focus on **integration**, not functioning of individual units
- Difficulty of **isolating defects** increases with integration scope → incremental/big bang integration

Test Levels – Unit vs. Integration Testing

Why not skip integration testing when unit tests worked perfectly fine?



Test Levels – System Testing

- **Definition**
  - Testing conducted on a complete, integrated system to evaluate the system‘s compliance with its specified requirements. (IEEE 610.12-90)

- Concerned with the behavior of a **whole system/product (end-to-end)**
- Test environment should mirror the production environment to minimize risk of missing **environment-specific errors**
- Often conducted by independent test team
- Often basis for release decisions

Test Levels – Acceptance Testing

- **Definition**
  - Formal testing conducted to determine whether or not a system satisfies its acceptance criteria and to enable the user, customer or other authorized entity to determine whether or not to accept a system. (IEEE 610.12-90)

- Establish **confidence** in the functional or non-functional characteristics of a system
- Evaluate the system‘s **readyness for delivery**
- Involves stakeholders such as users or customers of a system
- Uncovering defects is **not** the main purpose

Test Levels

What is the base for test specification?

What are we testing?

Test Level	Test Basis	Test Objects
Unit Testing	Requirements Detailed design Code	Components Data migration programs Database modules
Integration Testing	Software/System design Architecture Workflows Use Cases	Subsystems Infrastructure Interfaces System configuration
System Testing	System requirements Use cases Functional specification Risk analysis reports	System User and operation manuals System configuration
Acceptance Testing	User requirements System requirements Use cases Business processes Risk analysis reports	Fully integrated system Operational and maintenance processes User procedures Business processes Configuration data

Mutation Testing

- **How good are my tests?**

- Modify small pieces in the source code in order to seal artificial defects
- Analyse if existing unit tests are able to identify the defects
  - If not, regression testing would not be able to identify newly introduced problems
- Indicates the quality of the tests

Mutation Testing

1. Mutate the Code

- Mutate Logical Operators
  - if(a && b) → if(a || b)
- Mutate Boundaries
  - if(a > 10) → if(a > 11)
- Mutate Conditional Boundaries
  - if(a > b) → if(a >= b)
  - if(a < b) → if(a <= b)
- Negate Conditionals
  - if(a == b) → if (a != b)
- Mutate Math Operators
  - a + b → a * b
- …

2. Run existing Tests

Test(s) fail = mutants killed

- High code coverage
- High test quality
- Good assertions

Test(s) still green = mutants lived

- Mutated code is poorly/not covered by tests
  - → increase test coverage
- Quality of existing tests/test data needs to be improved

Mutation Testing – Example

```java
package at.ac.tuwien.inso.peso;

class DiscountHelper {

    double calculateDiscount(int age) {
        if(age < 18) {
            return 0.5;
        }

        return 1;
    }
}
```

```java
public class DiscountHelperTest {


    @Test
    public void test_shouldReturnDiscount50Percent() {
        DiscountHelper helper = new DiscountHelper();
        double discount = helper.calculateDiscount( age: 15);
        Assert.assertEquals(discount,  o1: 0.5);
    }

    @Test
    public void test_shouldReturnNoDiscount() {
        DiscountHelper helper = new DiscountHelper();
        double discount = helper.calculateDiscount( age: 35);
        Assert.assertEquals(discount,  o1: 1.0);
    }

}
```

Element	Class, %	Method, %	Line, %
DiscountHelper	100% (1/1)	100% (1/1)	100% (4/4)

DiscountHelper.java

```
1   package at.ac.tuwien.inso.peso;
2
3   public class DiscountHelper {
4
5       public double calculateDiscount(int age) {
6  2        if(age < 18) {
7  1            return 0.5;
8           }
9
10 1        return 1;
11      }
12  }
```

Mutations

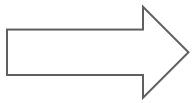
```
6   1. changed conditional boundary → SURVIVED
    2. negated conditional → KILLED
7   1. replaced return of double value with -(x + 1) for at/ac/tuwien/inso/peso/DiscountHelper::calculateDiscount → KILLED
10  1. replaced return of double value with -(x + 1) for at/ac/tuwien/inso/peso/DiscountHelper::calculateDiscount → KILLED
```

Extreme Mutation Testing

- **Approach Extreme Mutation**
  - Duration of mutation test execution increases dramatically
  - In Extreme Mutation testing, the whole logic of a method under test is eliminated

```java
public class Calculator {
    int result = 0;

    public void calculate(int[] numbers) {
        for (int number : numbers) {
            result += number;
        }
    }
}
```



```java
public class Calculator {
    int result = 0;

    public void calculate(int[] numbers) { }
}
```

  - Result:
    - pseudo-tested methods (e.g. tests without assertions) are located

Content

Test Types

- Focus on **a particular objective**
- Can be applied on **multiple test levels**
- Address specific **characteristics** of a system

- **Classified into:**

1. Functional Testing
2. Non-Functional Testing
3. Structural Testing
4. Change-Related Testing
    a. Confirmation Testing (Re-Testing)
    b. Regression Testing

Test Types – 1. Functional Testing

- **Functionality**
  - The capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions (ISO/IEC 25000)
- Functional tests are based on the **functions** and **features** of a system or component (ISTQB)
- Functional tests evaluate that a system or component complies with its **functional requirements**
- Rely on the **external** (input/output) **behavior** of the system → **Black–Box** design techniques → Lecture 3

- Focus: **WHAT** does the system do?

Test Types – 2. Non-Functional Testing

- Concerned with testing **quality attributes** of a system or software unit that do **not relate to functionality**
- Focus on **characteristics** of a system that can be **quantified** on a varying scale (ISTQB)
- **Non-Functional Testing includes:**
  - Performance Testing (e.g. response time)
  - Load Testing
  - Stress Testing
  - Usability Testing
  - …
- Late discovery of non-functional defects → major project risk
- Focus: **HOW WELL** does the system work?

Test Types – 2.1 Performance Testing

- **Performance**
  - Degree to which a system accomplishes functions within given constraints regarding processing time and throughput rate. (ISTQB)
  - Performance requirements examples:
    - *The system must respond within 2 seconds for 90% of the time*
    - *The system must manage up to 1000 concurrent users*
- **Performance Testing**
  - Testing the ability of a system or software unit to respond to inputs within a specified time and under specified conditions.

Test Types – 2.2 Load/Stress Testing

Load Testing

- Evaluate the behavior of a system or component with **increasing load,** e.g.
  - # of concurrent users
  - # of concurrent transactions
- Test the ability of the system to **handle increasing levels** of **anticipated realistic load**

Stress Testing

- Evaluate the behavior of a system or component
  - **at or beyond the limits** of its anticipated or specified load
  - or with **reduced availability of resources** (memory, server)
- Test the ability of the system to handle **peak loads**

Test Types – 2.2 Load/Stress Testing

- **Example:**
  A web system is – according to its specification – designed to handle 100 concurrent users at a time

- **Load Testing:**
  - Analyse how the system handles increasing levels of a realistic number of concurrent users, e.g. 20, 50, 75 users

- **Stress Testing:**
  - Analyse how the system handles 150 or 300 concurrent users

Test Types – 2.3 Usability Testing

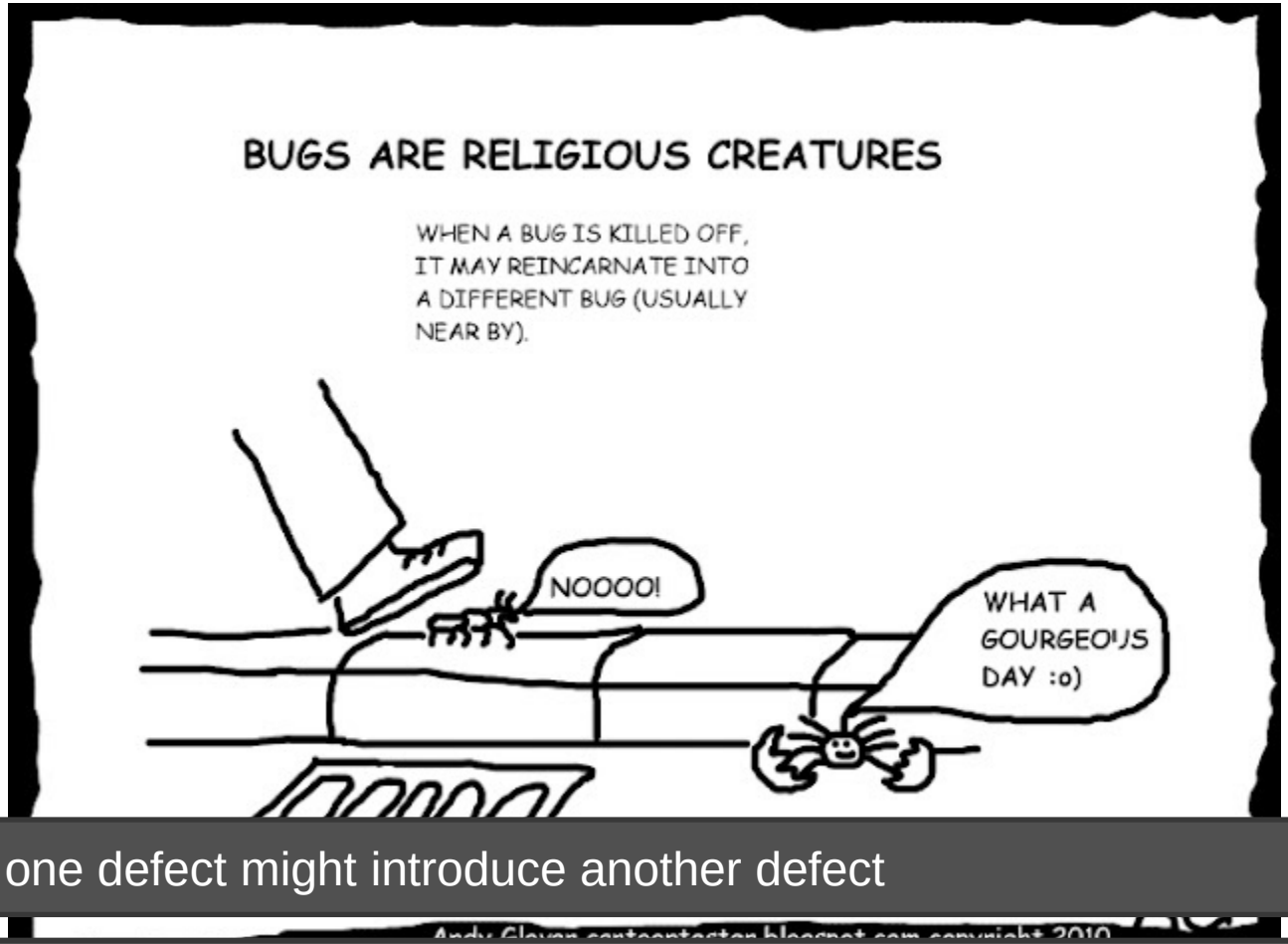
- Evaluate the degree to which a system can be used with **effectiveness**, **efficiency** and **satisfaction** within a specified context of use
- Test the capability of the software to be **understood, learned, used** by and it‘s **attractiveness** to the user

- **Types**
  - Exploratory Testing
    - Early in the development life cycle
    - Often with help of prototypes
    - Validate basic expectations and allow refinement
  - Validation Testing
    - Assessment of fully developed system

Test Types – 3. Structural Testing

- Based on the **internal structure, mechanism** and **architecture** of a system or component
- Also referred to as **white-box-testing** or **glass-box-testing**
- Focus on what happens **inside the application**
- Highly tool supported
- Includes assessing the extent to which a structure has been tested (**coverage**)
- Can be applied at all test levels
- Design techniques for structural testing
  - → Lecture 3

Test Types – 4. Change-Related Testing



1. Fixing one defect might introduce another defect
2. Fixing one defect might uncover another already present defect
3. Changing an existing part of a system might have unexpected side effects

Test Types – 4. Change-Related Testing

Confirmation Testing

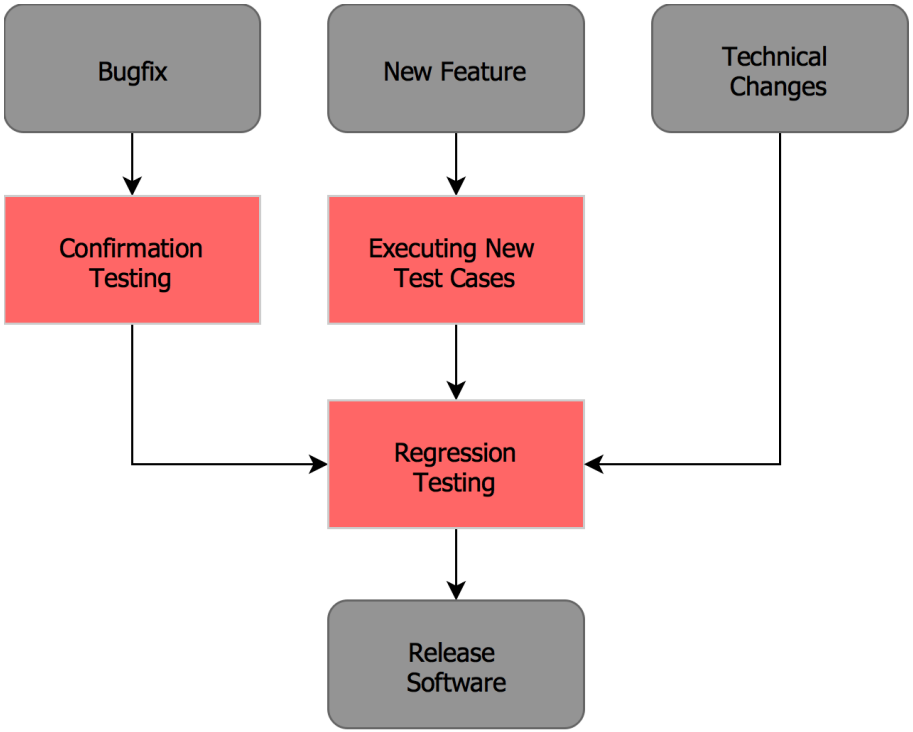
- Also referred to as **re-testing**
- Confirm that a **previously found defect** was **successfully fixed**
- Execute test cases that have **failed** in the **previous test run**

Regression Testing

- Repeated testing of a software after modification
- Ensure that **modifications** to the software **did not introduce** any new defects/side effects to already **existing features**
- Performed when software or its environment **changed**
- Strong indication for **automation**
- Applied in **all test levels** and for **all test types**

Test Types – 4. Change-Related Testing

- **Regression Testing**
  - Every change to a software can have **unexpected side effects**
  - **Changes include**:
    - Fixing defects
    - Implementing features
    - Technical changes
      e.g. refactorings,
      library upgrades, …
  - Scope of regression testing is determined **risk-based** depending on changes



Content

Maintenance Testing

- **Maintenance**
  - The process of modifying a software, system or component to correct faults, improve performance or other attributes or adapt to a changed environment. (IEEE 610.12-90)

- **Maintenance Testing**
  - Testing an existing operational system after
    - **Modification**
    - **Migration**
    - **Retirement**

Maintenance Testing

- **Required for**
  - **Enhancement** changes (i.e. planned release)
  - **Corrective/emergency** changes (i.e. hotfix release)
  - **Environmental** changes (OS/DB upgrades, …)
  - **Migrations** (between platforms, between data formats, …)

- **Regression** testing is an important part of maintenance testing

- **Challenges**
  - Outdated/missing specification
  - Missing domain knowledge

Content

Testcase

- **Definition**
  - A test case is a set of input values, pre-conditions, expected results and post-conditions, developed for a particular objective, such as to exercise a **particular program path** or to **verify compliance** with a **specific requirement.** (ISTQB)

- Each test case covers a single use case
- Each test case defines a step-by-step procedure
- Test cases need to be formally specified
- Test cases consist of specific attributes

Testcase Attributes

1. Test Case Specification Identifier (ID)
2. Test Case Description
3. Test Items
4. Test Procedure
5. Input Specification
6. Output Specification
7. Environmental Needs (opt)
8. Intercase Dependencies (opt)
9. Other Attributes (opt)

Testcase Attributes

1. **Test Case Specification Identifier (ID)**
   - Specify a unique generated number to identify this test case
   - Should follow a naming convention
   - Examples:
     - „TC_Login_001“
     - „TC_1003“

ID	TC_Search_001
Description	
Test Items	
Requirement	
Steps	
Input	
Expected Output	
Precondition	
Type	
Status	
Priority	

Testcase Attributes

2. Test Case Description

- Specify a quick summary (the goal) of this test case
- Examples:
  - „Test login with a wrong password“
  - „Test login with missing username“

ID	TC_Search_001
Description	Search for an existing article
Test Items	
Requirement	
Steps	
Input	
Expected Output	
Precondition	
Type	
Status	
Priority	

Testcase Attributes

3. Test Items

- Identify and briefly describe the items/features that should be covered by this test case
- Reference the source documents (Traceability)
  - Requirements specification
  - Design documents
  - …
- Example
  - „1512_Login Mask“

ID	TC_Search_001
Description	**Search for an existing article**
Test Items	Feature Item_Search
Requirement	RQ_1521
Steps	
Input	
Expected Output	
Precondition	
Type	
Status	
Priority	

Testcase Attributes

4. Test Procedure

- Describe the steps that are performed during test case execution

ID	TC_Search_001
Description	**Search for an existing article**
Test Items	Feature Item_Search
Requirement	RQ_1521
Steps	1. Navigate to the "Search" page 2. Enter the search text in the "Search" field 3. Submit by clicking button "Search"
Input	
Expected Output	
Precondition	
Type	
Status	
Priority	

Testcase Attributes

5. Input Specification

- Declare input values required to execute this test case
- Specify all required relationships beween inputs
- Values, messages, constants, human actions, files, …

ID	TC_Search_001
Description	**Search for an existing article**
Test Items	Feature Item_Search
Requirement	RQ_1521
Steps	1. Navigate to the "Search" page 2. Enter the search text in the "Search" field 3. Submit by clicking button "Search"
Input	Search Text: "Foundations of Software Testing"
Expected Output	
Precondition	
Type	
Status	
Priority	

Testcase Attributes

6. Output Specification

- Identify the expected outputs to verify this test case
- Define the expected changes of the system state
- Provide the exact value for each output
- Values, messages, files, response times, duration, …

ID	TC_Search_001
Description	**Search for an existing article**
Test Items	Feature Item_Search
Requirement	RQ_1521
Steps	1. Navigate to the "Search" page 2. Enter the search text in the "Search" field 3. Submit by clicking button "Search"
Input	Search Text: "Foundations of Software Testing"
Expected Output	1. Exactly one search result is displayed on the result page 2. Result contains the following information: 2.1. Title: "Foundations of Software Testing: ISTQB" 2.2. Price: "22.5€"
Precondition	
Type	
Status	
Priority	

Testcase Attributes

7. Environmental Needs (optional)

- Specify the characteristics and configurations of the hardware and software required to execute this test case
- Define pre-conditions of the test items
- Define special hardware installation (e.g. scanner/printer)

ID	TC_Search_001
Description	**Search for an existing article**
Test Items	Feature Item_Search
Requirement	RQ_1521
Steps	1. Navigate to the "Search" page 2. Enter the search text in the "Search" field 3. Submit by clicking button "Search"
Input	Search Text: "Foundations of Software Testing"
Expected Output	1. Exactly one search result is displayed on the result page 2. Result contains the following information: 2.1. Title: "Foundations of Software Testing: ISTQB" 2.2. Price: "22.5€"
Precondition	Exactly one article with title "Foundations of Software Testing: ISTQB" exists in the database
Type	
Status	
Priority	

Testcase Attributes

9. Other Attributes (optional)

- Type (Manual/Automated)
- Status (e.g. in work, reviewed, approved, deprecated)

ID	TC_Search_001
Description	**Search for an existing article**
Test Items	Feature Item_Search
Requirement	RQ_1521
Steps	1. Navigate to the "Search" page 2. Enter the search text in the "Search" field 3. Submit by clicking button "Search"
Input	Search Text: "Foundations of Software Testing"
Expected Output	1. Exactly one search result is displayed on the result page 2. Result contains the following information: 2.1. Title: "Foundations of Software Testing: ISTQB" 2.2. Price: "22.5€"
Precondition	Exactly one article with title "Foundations of Software Testing: ISTQB" exists in the database
Type	Manual
Status	Reviewed
Priority	2 - Important

Content

Defect Workflow

- **Defect**
  - **Discrepancy** between **expected** and **actual result**

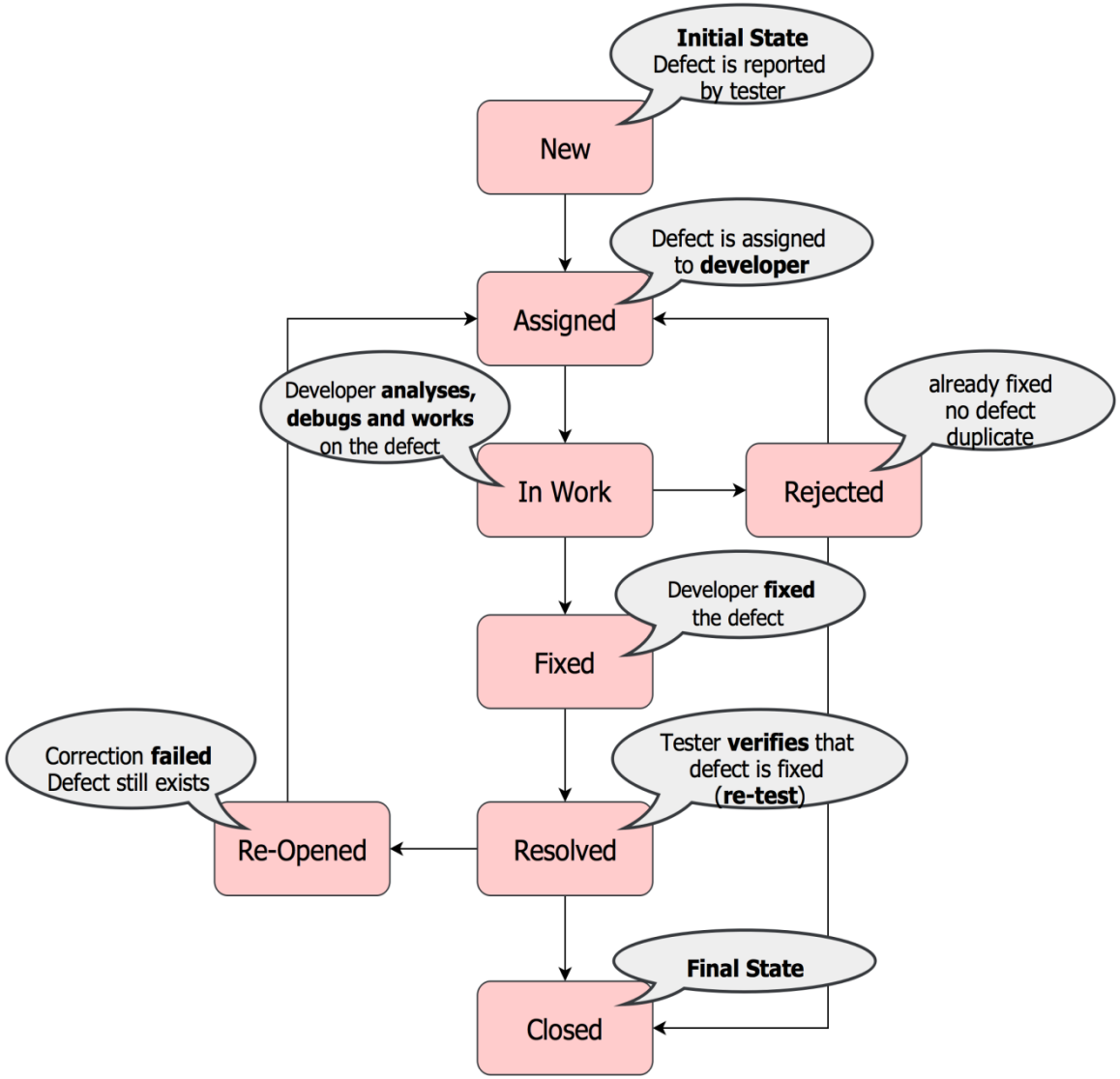
- **Defect Workflow**
  - Defines the life cycle of a defect from its **creation** to its **closure**
  - Gathers defect **information** and **status**
  - Allows to **monitor** and **control** release contents

  - Usually tool supported (e.g. JIRA, HPQC, Mantis, Redmine, Gitlab …)

Defect Workflow

- **Why do we need a defect management tool with a defined workflow?**
  - What is the overall state of a system?
  - How many defects are in a certain version?
  - What is the state of those defects?
  - Which developer is working on a defect?
  - Which tester has found that defect?
  - Was a problem already reported?
  - In which version has something been found/fixed?
  - Can we go live to production?

Defect Workflow – Sample Workflow



- Defect workflow can be **adopted** to the software project and may vary in **naming**
- Defect management is a **cross-functional team** task (test, development, project/product management)

Defect Report

- **Gathers information** on a detected defect
- Needs to be **complete**, **precise, objective** and cannot be replaced by informal communication

- **Includes the following information**
  - Person who discovered the defect
  - Current owner (person currently working on the defect)
  - Defect summary
  - Detailed defect description
  - Steps to reproduce the failure (including screenshots/logs/…)
  - Actual/Expected results
  - Severity / Priority
  - Environment
  - Reference to other work products (test case/requirement id)

Defect Report – Severity

- Defect severity is the **degree of impact** that a **defect** has on the **development/operation** of a system
  - Financial damage, harm, user satisfaction, reputation

- **Frequent classification**

  - *Critical* – core feature, critical data affected, no workaround, severe failure
  - *Major* – important feature affected, complex workaround exists
  - *Medium* – minor functionality affected, simple workaround exists
  - *Minor* – functionality not/slightly affected, inconveniece, styling/spelling issue

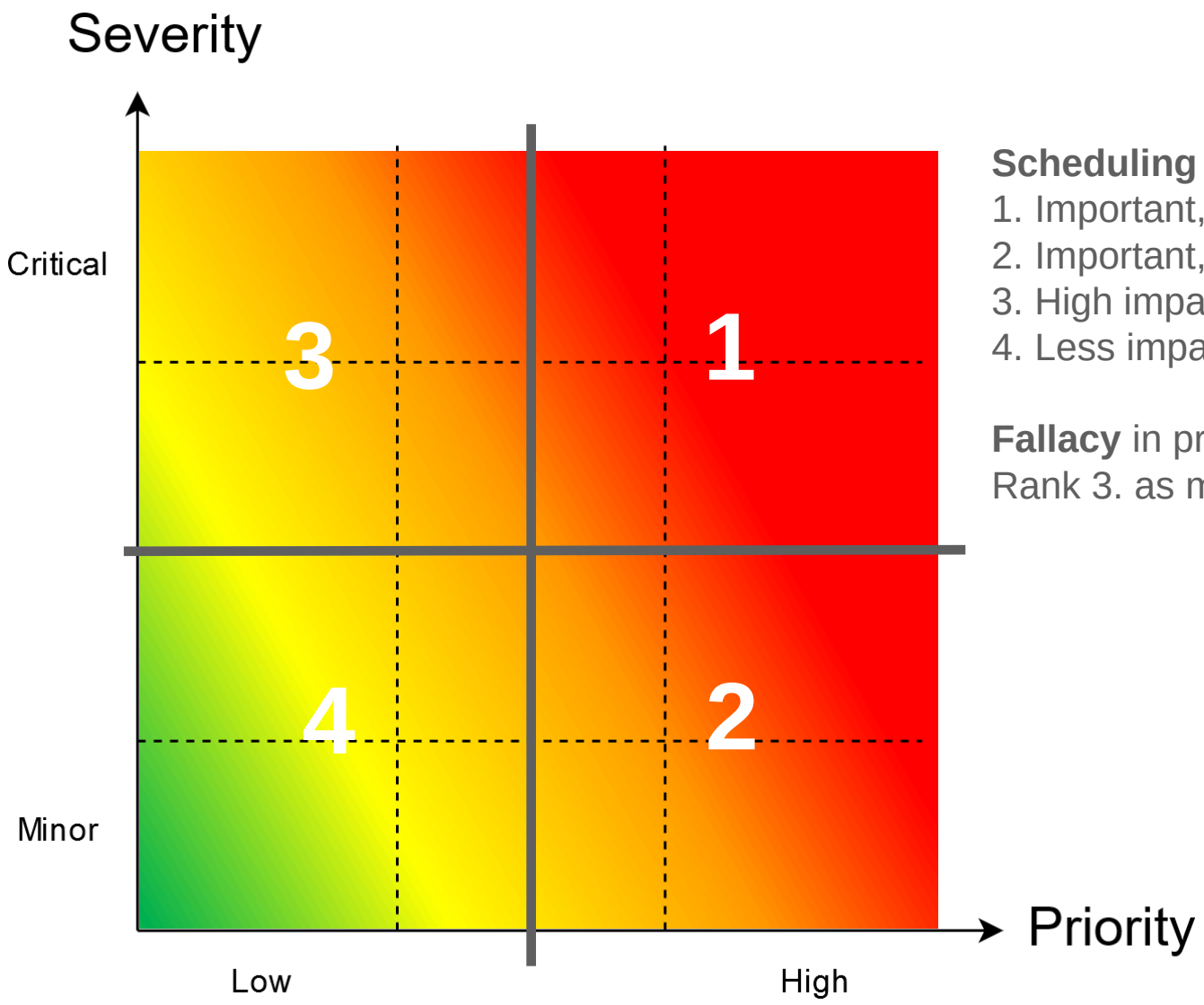
- Terminology depends on **project, organization** or **tool**

Defect Report – Priority

- Defect priority is the **level of (business) importance**
- Corresponds to the urgency a defect must be addressed
- Highly relevant for scheduling and release planning

- **Frequent classification**
  - *High* – important core use case, highly frequented by users, blocks users from proceeding their work
    → immediate fix required
  - *Medium* – important, still regularly used feature, but not ultimately blocking
    → fix in next release cycle
  - *Low* – less important use case, feature with small user base
    → fix can be deferred

Defect Report – Severity vs. Priority



Scheduling
1. Important, critical impact
2. Important, minor impact
3. High impact, less important
4. Less impact, less important

Fallacy in practice:
Rank 3. as most important

Defect Report – Severity vs. Priority Examples

Example 1:

„The wrong company logo was embedded in the application header“

- **Priority: high**
  - Negative impact on company image and reputation
- **Severity: minor**
  - Functionality is not affected
  - System does not crash

Defect Report – Severity vs. Priority Examples

Example 2:

„The feature annual statistics report export for application administrators crashes with a NPE“

- **Priority: low**
  - Affects a very small user group
  - Feature is not highly important for core business
- **Severity: major**
  - Functionality is affected, system might be in a wrong state
  - There exists no workaround (but feature is not ultimately critical)

Defect Report – Severity

Defect Report – Example

SWT Template Project / SWTORIGIN-2

Discount not accepted after entering promotion code

Details

Type:	Defect	Status:	NEW (View Workflow)
Priority:	↑ High	Resolution:	Unresolved
Severity:	↑ High		
Labels:	None		
Detected in Version:	1.0.1		
Requirement Id:	R3301		
Test Case Id:	22		

People

Assignee:	Unassigned
	Assign to me
Reporter:	Markus Zoffi
Votes:	0
Watchers:	1 Stop watching this issue

Description

After entering a valid promotion code for a 10% discount, the displayed overall price has not changed.

Steps:

1. Login with testuser 'tuser1'
2. Add item "Bram Stoker - Dracula" to cart
3. Click on "purchase"
4. Enter promotion code "TestPromo10"

Expected Result

The discount is granted for the overall price.
The overall price should reduce by 10%. (15,00 - 10% = 13,5)

Actual Result

The discount is NOT granted for the overall price.
The overall price remains the same.(15,00) as shown in screenshot

Dates

Created:	18 minutes ago
Updated:	3 minutes ago

Attachments

Drop files to attach, or browse.

Defect Report – Best Practices

- **Defects should be considered atomic**
  - One defect should only cover one problem
  - Raise another defect in case you identified additional problems
- **Defects should be self-contained**
  - By reading the defect, it should be possible to understand and reproduce the problem
- Reproduce the bug before reporting a defect
- Create links (references) between defects that are connected
- Include screenshots, but do not reduce bug reports to just adding pictures
- Update the status to enhance efficiency and clean reporting
- Use comments to make decisions/updates transparent

References

- T. Grechenig et al; Softwaretechnik Mit Fallbeispielen aus realen Entwicklungsprojekten, 2010
- ISTQB Foundation Level Syllabus
- A. Spillner and T. Linz; Basiswissen Softwaretest. 5. Aufl. dpunkt Verlag, 2012.
- D. Graham et al; Foundations of Software Testing: ISTQB Certification. Cengage Learning EMEA, 2008.
- IEEE Std 610.12-1990 (R2002), IEEE Standard Glossary of Software Engineering Terminology IEEE, 1990.
- A. Spillner et al; Praxiswissen Softwaretest – Testmanagement: Aus- und Weiterbildung zum Certified Tester, 2014
- R. C. Martin; Clean Architecture – A Craftsman‘s Guide to Software Structure and Design, Prentice Hall, 2018