This document has been elaborated for the project FITPED (https://www.fitped.eu)

FITPED

Work-Based Learning in Future
IT Professionals Education
(Grant. no. 2018-1-SK01-KA203-046382)

Co-funded by the
Erasmus+ Programme
of the European Union

Testing

2020

This project has been funded with support from the European Commission under the ERASMUS+ Programme 2018, KA2, project number: 2018-1-SK01-KA203-046382.

Content

Introduction to Software Testing

Chapter 1

1.1 Basic Testing Methods

🕮 1.1.1

Software testing is

- part of a more general verification and validation process, which also includes static validation techniques,
- the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not,
- executing a system to identify any gaps, errors, or missing requirements contrary to the actual requirements,
- intended to show that a program does what it is intended to do and to discover program defects before it is put into use.
- A process of analysing a software item to detect the differences between existing and required conditions and to evaluate the features of the software item.

Testing software often requires executing a program using artificial data. Testing can reveal the presence of errors, not their absence.

🕮 1.1.2

Testing is a part of all kinds of software development life cycles. Different types of stakeholders are involved in the process of software testing depending on the complexity of the project, used methodology and project management, the experience of the project team members. The following roles are often involved in the process:

- Software Tester,
- Software Developer,
- Project Manager,
- End-User,
- Software Quality Assurance Engineer,
- Quality Assurance Analyst.

🕮 1.1.3

Software Development Life Cycle (SDLC) often called the Software Development Process, is a process used by the software industry to design, develop and test high-quality software.

The SDLC aims to produce high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates. It has the following phases in general:

- Planning and requirements analysis
- Defining requirements
- Designing the product architecture
- Building the product
- Testing the product
- Deployment of the product
- Maintenance and further development of the product

🕮 1.1.4

There are numerous SDLC models, which are often called Software Development Process Models. They are suitable for different situations. Their effective use depends on the complexity of the software product, which should be developed. They differ in the series of steps, which ensure success in the process of software development. The following are the most popular SDLC models:

- Waterfall model
- Iterative model
- Spiral model
- V-model
- Agile model
- Rapid Application Development Model
- Prototyping model.

The software testing is included in all SDLC models in different extend. It can create a separate phase, or testing can create an inseparable part of different phases of the SDLC.

🕮 1.1.5

Verification and Validation are two very similar terms, which closely relate to the software testing topic. They differ in the following aspects:

Verification:

1. "Are you building it right?"
2. Ensures that the software system meets all the functionality.
3. Verification takes place first and includes the checking for documentation, code, etc.
4. It is done by developers.
5. It has static activities, as it includes collecting reviews, walkthroughs, and inspections to verify software.
6. It is an objective process, and no subjective decision should be needed to verify software.

Validation:

1. Validation addresses the concern: "Are you building the right thing?"
2. Ensures that the functionalities meet the intended behaviour.
3. Validation occurs after verification and mainly involves the checking of the overall product.
4. It is done by testers.
5. It has dynamic activities, as it includes executing the software against the requirements.
6. It is a subjective process and involves subjective decisions on how well the software works.

🕮 1.1.6

When to Start Testing?

Testing should start as early as possible because early start to testing reduces cost and time to rework the product. Real start depends on the used development model. Testing is incorporated in every phase of SDLC:

- During the requirement gathering phase, the analysis and verification of requirements are also considered as testing.
- Reviewing the design in the design phase with the intent to improve the design is also considered as testing.
- Testing performed by a developer on completion of the code is also categorised as testing.

🕮 1.1.7

When to Finish Testing?

Any software can not be 100% tested. Testing is a never-ending process. Therefore, it is important to estimate how much testing is enough and consider the following circumstances:

- Deadline of the project
- Code coverage level of the source code
- Bug rate under a certain level
- Decisions of the project manager

🕮 1.1.8

Advantages of Software Testing

- Cost-effectiveness – early testing saves time and costs because the discovered problem does not affect the final implemented solution
- Software improvement – testing is a phase of all SDLC
- Automation reduces the testing time. However, it should be started after static validation, an inspection of the system.
- Software quality assurance helps to measure the following set of software properties like functionality, reliability, usability, efficiency, maintainability, portability.

🕮 1.1.9

Program Testing has the following parts

Validation testing demonstrates to the developer and the system customer that the software meets its requirements. A successful test shows that the system operates as intended.

Defect testing, which discovers situations in which the behaviour of the software is incorrect, undesirable or does not conform to its specification. It leads to defect testing. A successful test is a test that force the system to perform incorrectly and so shows a defect in the system

🕮 1.1.10

Software inspection is a formal evaluation technique in which software requirements, designs, or codes are examined in detail by a person or a group other than the author to detect faults, violations of development standards, and other problems Complementary verification technique to testing. It represents a formal technique that involves formal or informal technical reviews of any artefact by identifying any error or gap. Software inspection is considered an effective approach for discovering program errors. It is a static verification because it focuses on the analysis of the static system representation to discover problems.Simultaneously, it does not require the execution of a system and, therefore, t does not require additional costs for inspection of incomplete versions of a system. It can be applied to any software representations like the requirements specification, software architecture, database schema. Software inspection can consider a broader set of quality attributes like portability, maintainability, compliance with standards. It can check conformance with a specification, not with the customers' requirements. On the other hand, it does not check non-functional requirements like performance, usability.

🕮 1.1.11

The software engineering defines the following two types of testing:

Manual testing covers testing a software manually, without any automated tool and scripts. Manual testing has several stages, which will be introduced later. The tester in the role of end-user test the software with the aim to identify any unexpected behaviour. The tester uses test cases and test scenarios to ensure the completeness of tests.

Automated testing (Test automation) requires the tester writes scripts and use specialised software to test the software product. Test automation is a logical replacement of the manual testing, in which the constantly repeating routines occur. It allows running the test scenarios repeatedly and incrementally.

🕮 1.1.12

When to Automate?

Test Automation is useful in the following situations:

- project is large and complex,
- projects require testing the same areas repeatedly,
- requirements do not change very often.

📖 1.1.13

There are many specialised systems, which allow automated testing. The following methodology can be used to decide if automated testing can be used:

- Identifying areas within the software for automation
- Selection of appropriate tool for test automation
- Writing test scripts
- Development of test suits
- Execution of scripts
- Create result reports
- Identify any potential bug or performance issues

📝 1.1.14

Mark the correct statements about software verification.

- Verification answers the question of whether we developed the software correctly.
- Verification answers the question of whether we have developed the right software.
- Verification is done by software developers.
- Verification is done by end users.
- Verification is an objective process that requires no subjective decisions.
- Verification is a subjective process that assumes subjective decisions about how software works properly.
- Verification precedes software validation.
- Verification follows software validation.

1.1.15

Mark the correct statements about software validation.

- Validation answers the question of whether we developed the software correctly.
- Validation answers the question of whether we have developed the right software.
- Validation is done by software developers.
- Validation is done by end users.
- Validation is an objective process that requires no subjective decisions.
- Validation is a subjective process that assumes subjective decisions about how software works properly.
- Validation precedes software validation.
- Validation follows software validation.

1.1.16

Which of the following test types can be automated?

- unit tests
- integration test
- performance tests
- functional tests
- usability tests
- acceptance tests

1.1.17

The aim of testing is to ensure 100% error-free software.

- false
- true

1.1.18

System testing is the level of software testing in which we test complete software using the black box method.

- true
- false

📝 1.1.19

Usability testing is a type of testing that is done from a developer perspective to see if the software is easy to use.

- false
- true

📝 1.1.20

Regression testing is a type of testing, in which we want to see if the changes we made to the source code have not retroactively affected the functionality of previously developed parts of the software.

- true
- false

1.2 Types of Testing

🕮 1.2.1

There are several types of testing:

- Smoke Testing
- Functional Testing
- Non-functional Testing

They all can be used on different levels of testing, which will be introduced later.

📖 1.2.2

Smoke Testing

Smoke testing, often called build verification testing, is a type of software testing that comprises of a non-exhaustive set of tests, which try to ensure that the most important functions of the software will work. The result of this testing is used to decide if a build is stable enough to proceed with further testing. This type of testing can uncover problems early. It can be used in integration, system and acceptance levels of testing.

📖 1.2.3

Functional Testing

Functional testing is a type of black-box testing. The software is tested using a set of tests with known inputs. The obtained outputs (results) are compared with expected ones. Functional testing has the following steps:

- The determination of the functionality that the intended application is meant to perform.
- The creation of test data based on the specifications of the application.
- The output based on the test data and the specifications of the application.
- The writing of test scenarios and the execution of test cases.
- The comparison of actual and expected results based on the executed test cases.

📖 1.2.4

Non-functional Testing

Non-functional testing involves testing important software non-functional requirements such as

- performance,
- security,
- user interface,
- compliance with standards.

Methods and Levels

Chapter 2

2.1 Testing Methods

🕮 2.1.1

Testing Methods

The following basic methods can be used based on the level of knowledge of the internal structure of the software, which is tested.

- Black-box testing
- White-box testing
- Grey-box testing
- Agile testing
- Ad-hoc testing

🕮 2.1.2

Black box Testing

Black box testing, known as behavioural testing, is testing without any knowledge of the internal structure, design or implementation. The tester has no access to the source code, but she interacts with the user interface of the software product. She provides a set of inputs and examines the outputs. The outputs must fulfil the tester's expectations. The main advantage of this approach is, that black-box testing is suitable for large code segments, does not require the access to source code, shows, how the software will be used by end-user, does not need testers with the knowledge of programming languages, operation systems and other whole SDLC. The main disadvantages of black-testing technique are limited coverage by tests, difficulties to design test cases and limited knowledge of the testers about the software product. This method attempts to find incorrect or missing functions, interface errors, errors in data structures, behaviour as well as performance. It is applicable to the integration, system and acceptance testing levels.

🕮 2.1.3

White box Testing

This testing method, also known as glass testing, requires the tester knows the internal structure, design or implementation of the software. In other words, she has access to the source code and can investigate the internal logic and structure of the code. This is simultaneously the main advantage of this method. Moreover, it allows code optimising, refactoring and the maximal coverage of the code due to the knowledge of the code. On the other hand, this technique requires skilled tester and specialised tools like code analyser and debugging tools. This method is applicable to unit, integration and system testing levels.

🕮 2.1.4

Grey box Testing

The grey box testing method has limited knowledge of the internal structure and logic of the tested software product. The tester usually must design documents and the database. Therefore, she can write better test scenarios. The combination of best practices of white box and black box methods is considered the main advantage of this method. It relies on interface definition and functional specifications. The tests are realised from the end-user's point of view, not a developer. It is primarily used in integration testing level.

🕮 2.1.5

Agile Testing

Agile testing represents a testing method, which follows the principles of agile development methods. This testing method does not require any special approach and techniques. It still needs all proven software testing methods and levels, but their use depends predominantly on the tester or developer decision and other priorities of the agile team. Agile testing is built upon very simple, strong and reasonable processes like the process of conducting the daily meeting or preparing the daily build. Simultaneously, it attempts to leverage tools, especially for test automation, as much as possible. Testing itself is in the middle of interest. As a result, this method does not elaborate on any plan or documentation.

2.1.6

Ad-hoc Testing

Ad-hoc testing, sometimes called as Random Testing or Monkey Testing, is a software testing method without any planning and documentation. All tests are conducted informally and randomly without any formal procedure or expected results. This method is normally used during Acceptance Testing. Surprisingly, this method can be very useful in finding errors, which is hard to find using other more systematic, step-by-step approach. The success of the method, therefore, depends on the creativity and previous experience of the tester.

2.1.7

Which level of testing tests the smallest part of the software?

- unit testing
- integration testing
- system testing
- debugging
- source code testing
- acceptance testing

2.1.8

Which testing method does the unite testing mostly use?

- white box
- black box
- brut force testing
- grey box
- yellow box
- divide et impera

2.1.9

Who participates in acceptance testing?

- end-users
- developer
- customer
- manager
- CIO
- researcher

2.1.10

Which level of testing use black box testing method?

- integration testing
- acceptance testing
- unit testing
- system testing

2.1.11

Which level of testing use white box testing method?

- integration testing
- acceptance testing
- unit testing
- system testing

2.2 Testing Levels

2.2.1

Unit Testing

Unit testing is a level of testing, where individual units/components are tested with the aim to verify that these units behave as expected. Unit testing belongs to the white-box testing method. A unit is the smallest tested part of the software, for example, method of a class in OOP. Unit testing is performed mainly by the developer, who can also be the author of the source code or other member of the development team. The developer uses test data. The

main aim is to validate that part of the source code is correct in terms of requirements and functionality. It is impossible to cover all source code and evaluate all possible execution paths of the software.

Unit testing has the following benefits:

- increase confidence in changing code,
- code is easier to reuse,
- development is faster,
- the cost of fixing a bug is smaller,
- small units are easier to understand.

🕮 2.2.2

Integration Testing

Integration testing means testing of combined parts of the software with the aim to determine if the parts work correctly. The purpose of this second level of testing is to expose faults in the interaction between integrated units. Integration testing can use bottom-up, top-down and big bang approach. While the first one begins with unit testing, followed by a combination of module testing and builds, the second one the modules are tested first and then the lower-level modules and units are tested. A big bang is an approach, where all or most of units are combined together and tested at one run. Integration testing uses any of black-box, white-box or grey-box testing methods.

🕮 2.2.3

System Testing

System testing is the third level of software testing. It tests the whole system to verify if it meets functional and technical specifications. After all the components are integrated, the software is tested with the aim to fulfil the specified quality standards. This kind of test enables to test, verify and validate not only business requirements but also the software

architecture in the environment, which is very close to the production environment. Black box testing method is usually used form system testing.

🕮 2.2.4

Acceptance Testing

Acceptance testing is closely joined to quality assurance. Acceptance testing is the fourth and last level of software testing. It verifies whether the software meets previously defined specifications and satisfies the business requirements of the customer. A system is tested for acceptability. Acceptance tests are intended to point out any bugs, which will result in the software crash, but it also points out small mistakes, errors and differences. It usually uses the black-box testing method. It does not follow a strict procedure and is rather ad-hoc.

🕮 2.2.5

Alpha Testing

Alpha test is realised by the development teams in the first stage of testing. It means that combined unit, integration and system testing are considered together as alpha testing. The software is tested for spelling mistakes, broken links.

🕮 2.2.6

Beta Testing

The beta testing (also called pre-release testing) follows alpha testing after it has been successfully finished. A selected group of future users tests the software. It is important to distribute the testing to a wide range of future users, who will test installation procedure, typography, navigation and flow of tasks, etc. Simultaneously, they provide important feedback, identify hidden problems and test their fixes.

Advanced Testing

Chapter 3

3.1 Non-functional Types of Testing

🕮 3.1.1

Regression Testing

All changes in the software can cause problems in other areas of the software. For that reason, the regression testing focuses on verification, if the change has not resulted in another functionality violation. In other words, regression testing ensures that this change has not caused problems, which are not covered by the tests. During regression testing, new test cases are not created but previously created test cases are re-executed. Regression testing can be used during any level of testing, mainly during system testing.

🕮 3.1.2

Usability Testing

Usability testing is a black-box technique used for identification of errors and consequent implementations of the software improvements by observing the users' behaviour. This testing is done from a user perspective with the aim to find out, if the software is easy to use. It is focused on the efficiency of use, ability to learn, ability to memorise, errors and safety and satisfaction of the users. This type of testing can be performed during system and acceptance testing levels.

🕮 3.1.3

Security Testing

Security testing belongs to the critical and inevitable kinds of non-functional testing. Its aim is to identify any problems with the security and vulnerability of the software. Depending on the nature of the software, security testing tries to ensure integrity, availability, correct authorisation and authentication, and save software against different kinds of attacks and flaws.

🕮 3.1.4

Portability Testing

Portability testing is focused on testing software for reusability, transferring software between computers and different versions of operating systems and middleware.

🕮 3.1.5

Compliance Testing

Compliance testing, sometimes called a conformance testing or regulation testing, is a type of testing to determine the compliance of a system with internal or external standards. The method and type of testing to be conducted during compliance testing depends on the specific regulation / standard being assessed.

🕮 3.1.6

Performance Testing

Performance testing is a type of non-functional testing focused on determining how a software performs in terms of responsiveness and stability under different conditions. It covers load tests, stress tests, endurance and spike tests. While Load testing test the behaviour of the software by application maximum load of input data, stress testing tests the behaviour under abnormal conditions like losing resources. Performance testing tries to identify any bottlenecks related to the software performance like network delay, load balancing between servers, database transaction delay using quantitative and qualitative measures. It tests speed, capacity, stability and scalability of the software.