**Tutorial. Software Testing**

V1.0

22/04/2020

In this tutorial we are going to cover the very basics of software testing. Below are the topics of this tutorial.

**Topics**

[**1.** **Software Development Life Cycle (SDLC)** 1](#_Toc101822857)

[**Class Activity** 4](#_Toc101822858)

[**2.** **Type of software test** 4](#_Toc101822859)

[**3.** **Software test process** 5](#_Toc101822860)

[**Step 1.** **Establish testing requirements** 5](#_Toc101822861)

[**Step 2.** **Prepare test plan document** 6](#_Toc101822862)

[**Step 3.** **Write auto test procedure** 6](#_Toc101822863)

[**Step 4.** **Perform test executions** 6](#_Toc101822864)

[**Step 5.** **Record test results and produce test report** 6](#_Toc101822865)

[**Step 6.** **Analysis test results** 6](#_Toc101822866)

[**Step 7.** **Manage software defects** 6](#_Toc101822867)

[**4.** **Software test principles guidelines** 6](#_Toc101822868)

[**5.** **Test-case design techniques** 7](#_Toc101822869)

[**Equivalence Partitioning (EP)** 7](#_Toc101822870)

[**Boundary Value Analysis (BVA)** 7](#_Toc101822871)

[**Negative Testing** 8](#_Toc101822872)

[**Decision Table Testing** 9](#_Toc101822873)

[**State Transition Diagrams** 9](#_Toc101822874)

[**6.** **Software change request management** 9](#_Toc101822875)

1. **Software Development Life Cycle (SDLC)**

In systems engineering, information systems and software engineering, the systems development life cycle (SDLC), also referred to as the application development life cycle, is a process for planning, creating, testing, and deploying an information system.

SDLC involves 6 stages:

* Determine Requirements
* Design
* Build
* Test
* Deploy
* Maintain

4 levels of software test

* Unit test – unit is the smallest piece of code that can be logically isolated in a system. E.g. a public method in a class. Unit test is a way of testing individual unit.
* Integration test – a type of software testing in which the different units, modules or components of a software application are tested as a combined entity (an integration of more than 1 unit but not whole system).
* System test – testing the system as a whole. All the units/modules/components are integrated in order to verify if the system works as expected or not.
* Acceptance test – once the system testing process is completed by the testing team and is signed-off, the entire product/application is handed over to the customer/few users of customers/both, to test for its acceptability.

Functional teams involve in the process

* **Architect/system design team** – Architect and system designer are the expert of the software product. They build the software architecture; they decide what features to be developed; they define how the whole system is spitted into components and the interfaces between the components; they also determine the technologies to be used.
* **Software development team** – Software developers build the software according to the design documents created by the architect and system designer and they run unit tests in a white-box manner. Also software developers work on the software change requests raised by the tester team to fix software defects (bugs).
* **Software test team** – Software testers develop test plans based on the requirement specification. And then they execute the test when the software modules or the whole product are ready to be tested. They produce test report and work with developer team to address software defects.
* **Software support** – While the above three teams work on the future releases, support people work on the software that has been delivered to the client. There are different levels of support, level 1 to 4, among which level 3 & 4 are responsible of maintaining the source codes of the software product, meaning people who work in level 3 & 4 are software developers. Level 3 & 4 support team works on fixing software defects reported from live customer. They need to do both coding and testing to build software patches and then deploy to the live sites.

In organization that develop large scale software, there are dedicated teams work in each functional team above. In organization that develop medium scale software, there might be only one development team and one test team. For small scale software, there might be no dedicated test team and software developers handle everything.

Refer to the following charts to understand software development life cycle. And how does software testing is involved.

**Determine Requirements**

**Design**

**Build**

**Test**

**Deploy**

**Maintain**

**Stages**

Requirement Docs

Doc

SW Design / Test Plan Doc

Untested Software

Tested Software

Online Software

Software Patches

**Outcome**

Architect / System Designer / Client

Architect / System Designer

Software Engineer

Software Tester

Deployment Engineer

Support Engineer

**By Whom**

Unit Test

System Test

Acceptance Test

All Levels of Tests Applied

**Software Test**

**Software Development Life Cycle**

Integration Test

Test Execution

Test Plan Development

**Class Activity**

Read this article - <https://stackify.com/what-is-sdlc/>

Watch this video - <https://www.linkedin.com/learning/software-development-life-cycle-sdlc/waterfall-model-brief-overview?autoplay=true&resume=false&u=57076585>

1. **Type of software test**

There are many types of software test. In fact, there are too many, check [this article](https://www.softwaretestinghelp.com/types-of-software-testing/). Use the following chart to build an overall understanding of different types of software test.

Unit Test

System Test

Acceptance Test

Integration Test

White Box

Black Box

Gray Box

* Unit Test
* Branch / Coverage Test
* Regression Test
* Boundary Value Test
* …
* Functional Test
* Regression Test
* End-to-End Test
* Alpha Test
* Balta Test
* …

1. **Software test process**

The following process are usually followed for any type of software test

* 1. **Establish testing requirements**

In this step, we need to list all the functional and non-functional requirements that need to be tested.

* Functional Requirements - are product features or functions that developers must implement to enable users to accomplish their tasks.
* Non-functional Requirements - define system attributes such as security, reliability, performance, maintainability, scalability, and usability.
  1. **Prepare test plan document**

Refer to attached test plan template file for more details. The core part of the test plan is the test cases. Some test cases are to be executed manually while some for them shall be executed automatically.

Also please refer to the “Test-case design techniques” section below for more details about designing test cases.

* 1. **Write auto test procedure**

If test cases are to be executed automatically, e.g., unit test, an auto test framework will be used for programming and run test cases.

“MSTest” is an auto test framework provided by Visual Studio. Refer to [this tutorial](https://docs.microsoft.com/en-us/visualstudio/test/getting-started-with-unit-testing) for more details.

* 1. **Perform test executions**

In this step software product or its units/components are ready to be tested. Testers will follow the test plan and run the test cases, manual or auto. During the procedure, all output need to be captured and store properly according to organizational process.

* 1. **Record test results and produce test report**

Based on the test output from last step, a formal report needs to be generated based on the test results, using organizational template (see attached “**Software\_Test\_Report\_Template.xlsx**” as an example).

* 1. **Analysis test results**

Test results and test report need to be effectively analysed by the developers to determine the cause of the failed the test cases.

* If the failure indicates software defects, source code shall be changed.
* If the failure indicates configuration problems, configuration procedure needs to be amended.
* If the test case itself has an issue, it needs to be resolved.
  1. **Manage software defects**

Every organization has its own procedure to manage software defect. Usually, a ticket system is used to manage software defects; and a “Change Requests (or CR)” is raised in the ticket system to track each defect.

Also please refer to the “Software change request management” section below for more details about CR management.

After all defects are resolved by developer team, tester will repeat step 4 ~ step 7 until the test achieve the completeness criterial.

1. **Software test principles guidelines**

Refer to attached “**Software\_Test\_Principles\_Guidelines.docx**” for more details

1. **Test-case design techniques**

**Equivalence Partitioning (EP)**

This technique divides the input data into different equivalence data classes. It can be applied where there is a range in the input field. Like below the input range is (-∞, +∞), which is divided into 4 partitions.

-∞

+∞

And the hypothesis behind this technique is that

* if one condition/value in a partition passes all others will also pass. AND
* if one condition in a partition fails, all other conditions in that partition will fail.

When applying technique to design test case, choose one value as input for each partition. For example, for the grading function, according to the requirement, the range of input mark value could be divided into the following partitions.

mark < 0

0<=mark < 60

60<=mark < 70

90<=mark < =100

mark > 100

……

And the following test cases can be developed to test the grading function, by using the **Equivalence Partitioning** technique.

|  |  |  |
| --- | --- | --- |
| **Partition** | **Test data** | **Expected Result** |
| Mark < 0 | -12 | invalid |
| 0 **≤** Mark < 60 | 36 | F |
| 60 **≤** Mark < 70 | 64 | D |
| 70 **≤** Mark < 80 | 77 | C |
| 80 **≤** Mark < 90 | 82 | B |
| 90 **≤** Mark **≤**100 | 98 | A |
| 100 < Mark | 120 | invalid |

**Boundary Value Analysis (BVA)**

Boundary testing is the process of testing between extreme ends or boundaries between partitions of the input values. So these extreme ends like Start- End, Lower- Upper, Maximum-Minimum, Just Inside-Just Outside values are called boundary values and the testing is called “boundary testing”.

The basic idea in normal boundary value testing is to select input variable values at their:

* Minimum
* Just above the minimum
* Just below the maximum
* Maximum

Notes:

* In Boundary Testing, Equivalence Class Partitioning plays a good role
* Boundary Testing comes after the Equivalence Class Partitioning.

For example, for the grading function, for each partition, the following input should be tested using this technique.

Min

Max

Min+

Max-

And the following test cases can be developed to test the grading function, by using the **Boundary Value** technique. (Note duplicate test cases are ~~crossed~~)

|  |  |  |
| --- | --- | --- |
| **Partition** | **Test data** | **Expected Result** |
| Mark < 0 | -1 | invalid |
| ~~0~~ | ~~F~~ |
| 0 **≤** Mark < 60 | 0 | F |
| 1 | F |
| 59 | F |
| ~~60~~ | ~~D~~ |
| 60 **≤** Mark < 70 | 60 | D |
| 61 | D |
| 69 | D |
| ~~70~~ | ~~C~~ |
| 70 **≤** Mark < 80 | 70 | C |
| 71 | C |
| 79 | C |
| ~~80~~ | ~~B~~ |
| 80 **≤** Mark < 90 | 80 | B |
| 81 | B |
| 89 | B |
| ~~90~~ | ~~A~~ |
| 90 **≤** Mark **≤**100 | 90 | A |
| 91 | A |
| 99 | A |
| 100 | A |
| 100 < Mark | ~~100~~ | ~~A~~ |
| 101 | invalid |

**Negative Testing**

The software testing can also be classified as two types of testing - positive testing and negative testing. Positive testing is the application functions as expected valid data sets. If an error occurs during positive testing the test fails. In the Negative Testing, the application handles the unexpected behaviour or invalid input. For example, if a user tries to type a wrong data in the field it would display a correct message for incorrect data. The quality of the application improves due to the negative testing.

In the grading function example, while **-10** is an invalid mark value, it is still considered as “positive” input, because -10 is a valid number and the function accept any number as input. However, value “**abc123**” is considered as “negative” input, because this value is not a number. With negative testing, an invalid input such as “**abc123**” will be used to develop a test case.

Read this article for more details about negative testing

<https://www.h2kinfosys.com/blog/negative-testing-3/>

**Decision Table Testing**

When dealing with more than one inputs, using decision table technique can help you to cover all necessary inputs combinations. Read this article to learn more about this technique

<https://www.merixstudio.com/blog/decision-table-software-testing/>

**State Transition Diagrams**

State Transition Testing Technique is helpful where you need to test different system transitions. E.g. user account will be blocked for 15 mins after 5 failed login attempt.

Read this article to learn more about this technique

<https://www.guru99.com/state-transition-testing.html>

1. **Software change request management**

As discussed above in the software test process section, whenever potential software defect is identified by the tester, a “Change Request” (or CR) shall be raised by the test and send to the developer for further investigation to resolve the issue.

CR are usually managed in a ticket system. Tester will create a CR in the system and fill in the following information

* A title
* Description of the issue
* Software version
* Testing configuration
* Testing steps to reproduce the issue
* Logs captured by the tester, showing the problem

The CR will then be assigned to a developer for investigation. Usually, the developer will take the following actions.

* Read the CR information and confirm with the tester if necessary
* Reproduce the issue following the steps provided
* Troubleshoot the source code to identify the root cause
* If software defect is identified, fix the error in the source code, and build a new software version
* Repeat the steps to verify the issue is cleared
* Fill in the following information in the CR
  + Root cause analysis
  + Resolution
  + Logs captured by the developer with the fix, showing that the problem is now fixed
  + New software version

The CR will then be assigned back to the tester. The tester will run the test again to verify the fix and close the CR.

In software development project, CR is usually tracked in the following ways:

**Scenario 1 - a general issue**

1. Programming team finished initial version of the software + documentation
   1. Baseline was built against current codes + documentation
2. Tester verified the functionality and found an issue (unexpected result)
3. Tester checked the setting up and test process and confirmed that this issue was potentially related to the coding
   1. The issue related to a **required core functionality**
4. Tester submitted a CR with the related log using **CR management system** - *CR status: "new"*
5. CR was evaluated by a **screener**
6. CR was categorized as "**Major**" - *CR status: "accepted"*
7. CR was sent to **lead programmer** for review - *CR status: "ready for review"*
8. CR was **approved** and **assigned** to programmer Joe to work on - *CR status: "assigned"*
9. Joe **investigated** the log and **reproduced** the issue in lab - *CR status: "in progress"*
10. Joe confirmed that the software has a **defect** (bug)
11. Joe updated the code and related documentation / verified the result - *CR status: "solution is ready"*
12. Software were sent to test team for verification - *CR status: "ready to verify"*
13. Tester verified the functionality and the result was successfully - *CR status: "verification passed"*
14. Testing team agreed to **close** this CR request - *CR status: "closed"*
15. Baseline was built against the latest codes + documentation, and labelled with the CR No.

**Scenario 2 - a emergency issue**

1. Programming team finished initial version of the software + documentation
   1. Baseline was built against current codes + documentation
2. Tester verified the functionality and found an issue (unexpected result)
3. Tester checked the setting up and test process and confirmed that this issue was potentially related to the coding
   1. The issue related to a **required core functionality**
   2. Rest of the test work has to be stopped until the issue is fixed
4. Issue was reported to project manager; an emergency conference call was hold
   1. In the meeting project manager, tester, **lead programmer,** programmer Joe had a discussion about the issue
   2. CR was created and was categorized as "**Emergency**" - *CR status: "new"*
5. CR was assigned to Joe in the meeting.
6. Joe started work on the investigation immediately - *CR status: "in progress"*
7. Joe confirmed that the software has a **defect** (bug)
8. The rest of story was the same as scenario 1, except the CR is treated as high priority. So the verification of this solution was scheduled above everything else.
   1. *CR status: "solution is ready" -> "ready to verify" -> "verification passed" -> "closed"*

End of this tutorial