



Learning Report – TDLC



L&T Technology Services



Document History

| Ver. Rel. No. | Release Date | Prepared. By | Reviewed By | Approved By | Remarks/Revision Details |
|---------------|--------------|--------------|-------------|-------------|--------------------------|
| 1.0 | 1/3/2021 | Sneha Anand | | PSP | |
| | | | | | |
| | | | | | |
| | | | | | |

Contents

| | |
|---|-----------|
| LIST OF FIGURES | 4 |
| 1. TDLC: TESTING DRIVEN LIFE CYCLE..... | 5 |
| PHASES IN STLC: | 6 |
| WHAT IS TESTING?..... | 7 |
| OBJECTIVES OF TESTING: | 7 |
| DIFFERENCE BETWEEN TESTING AND DEBUGGING..... | 7 |
| PRINCIPALS OF TESTING: | 8 |
| MODELS USED FOR TESTING: | 9 |
| LEVELS OF TESTING: | 13 |
| TYPES OF TESTING | 15 |
| TYPES OF TESTCASES: | 18 |
| 2.ACTIVITY: | 18 |
| ACTIVITY 1 –REQUIREMENTS ISSUE | 19 |
| ACTIVITY 2 – CHECKING CODE QUALITY | 19 |
| ACTIVITY 3 – CHECKING THE IMPLEMENTATION OF THE CODE..... | 19 |
| ACTIVITY 4 – SOLVING OF ISSUES..... | 19 |

LIST OF FIGURES

| | |
|--------------------------------------|-------------------------------------|
| Figure 1: TDLC cycle | 5 |
| Figure 2:waterfall model..... | Error! Bookmark not defined. |
| Figure 3:Agile Model..... | Error! Bookmark not defined. |
| Figure 4:levels of testing | 13 |
| Figure 5:functional testing | 15 |
| Figure 6:non-functional testing..... | Error! Bookmark not defined. |

1. TDLC: Testing Driven Life Cycle

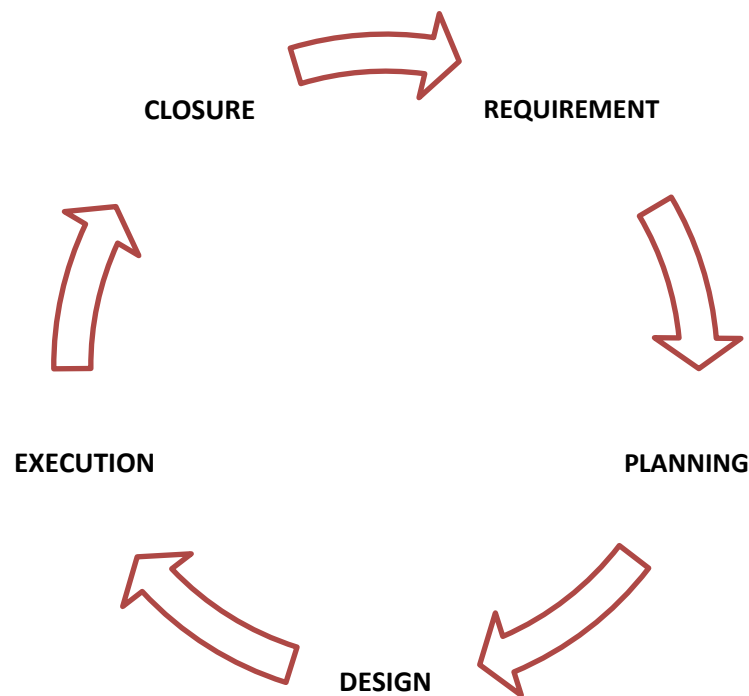


Figure 1: TDLC cycle

Software Testing Life Cycle (STLC) is a sequence of different activities performed during the software testing process.

In the initial stages of STLC, while the software product or the application is being developed, the testing team analyzes and defines the scope of testing, entry and exit criteria and the test cases. It helps to reduce the test cycle time and enhance the product quality.

As soon as the development phase is over, testing team is ready with test cases and start the execution. This helps in finding bugs in the early phase.

Characteristics of STLC:

- STLC is a fundamental part of Software Development Life Cycle (SDLC) but STLC consists of only the testing phases.
- STLC starts as soon as requirements are defined, or software requirement document is shared by stakeholders.
- STLC yields a step-by-step process to ensure quality software.

Phases in STLC:**1. Requirement phase:**

This phase helps to identify the scope of the testing. If any feature is not testable, communicate it during this phase so that the mitigation strategy can be planned.

2. Planning:

Test planning is the first step of the testing process. In this phase, we identify the activities and resources which would help to meet the testing objectives. During planning we also try to identify the metrics, the method of gathering and tracking those metrics.

3. Design Phase:

- i. Detail the test condition. Break down the test conditions into multiple sub-conditions to increase coverage.
- ii. Identify and get the test data.
- iii. Identify and set up the test environment.
- iv. Create the requirement traceability metrics.
- v. Create test coverage metrics.

4. Execution:

As the name suggests, this is the Software Testing Life Cycle phase where the actual execution takes place. But before you start your execution, make sure that your entry criterion is met. Execute the test cases, log defects in case of any discrepancy. Simultaneously fill your traceability metrics to track your progress.

5. Closure:

Tasks for the closure activities include the following:

- i. Check for the completion of the test. Whether all the test cases are executed or mitigated deliberately. Check there is no severity 1 defects opened.
- ii. Do lessons learned meeting and create lessons learned document (Include what went well, where are the scope of improvements and what can be improved)

What is Testing?

Software testing is a process, to evaluate the functionality of a software application with an intent to find whether the developed software met the specified requirements or not and to identify the defects to ensure that the product is defect-free in order to produce a quality product.

It involves execution of software/system components using manual or automated tools to evaluate one or more properties of interest. The purpose of software testing is to identify errors, gaps, or missing requirements in contrast to actual requirements.

Objectives of testing:

- a) To prevent defects by evaluate work products such as requirements, user stories, design, and code.
- b) To verify whether all specified requirements have been fulfilled.
- c) To check whether the test object is complete and validate if it works as the users and other stakeholders expect.
- d) To build confidence in the level of quality of the test object
- e) To find defects and failures thus reduce the level of risk of inadequate software quality.
- f) To provide sufficient information to stakeholders to allow them to make informed decisions, especially regarding the level of quality of the test object.
- g) To comply with contractual, legal, or regulatory requirements or standards, and/or to verify the test object's compliance with such requirements or standards.

Difference between testing and debugging.**TESTING**

- Testing is the process to find bugs and errors.
- It is the process to identify the failure of implemented code.
- Testing is the display of errors.
- It performed based on the testing type which we need to perform unit testing, integration testing, system testing, user acceptance testing, stress, load, performance testing etc.

DEBUGGING

- Debugging is the process to correct the bugs found during testing.
- It is the process to give the absolution to code failure.
- Debugging is a deductive process.
- Debugging is based on different types of bugs.

Principals of Testing:

- **Testing shows the presence of defects, not their absence.**
The goal of software testing is to make the software fail. Software testing reduces the presence of defects. Software testing talks about the presence of defects and does not talk about the absence of defects.
- **Exhaustive testing is impossible.**
Exhaustive testing is impossible means the software can never test at every test cases. It can test only some test cases and assume that software is correct, and it will produce the correct output in every test cases. If the software will test every test case then it will take more cost, effort, etc. and which is impractical.
- **Early testing saves time and money.**
To find the defect in the software, early test activity shall be started. The defect detected in early phases of SDLC will very less expensive.
- **Defects cluster together.**
In a project, a small number of the module can contain most of the defects. Pareto Principle to software testing state that 80% of software defect comes from 20% of modules.
- **Beware of the pesticide paradox.**
Repeating the same test cases again and again will not find new bugs. So it is necessary to review the test cases and add or update test cases to find new bugs.
- **Testing is context dependent.**
Testing approach depends on context of software developed. Different types of software need to perform different types of testing.
- **Absence-of-errors is a fallacy.**
If a built software is 99% bug-free but it does not follow the user requirement then it is unusable. It is not only necessary that software is 99% bug-free but it also mandatory to fulfil all the customer requirements.

Models used for Testing:

1. Waterfall model:

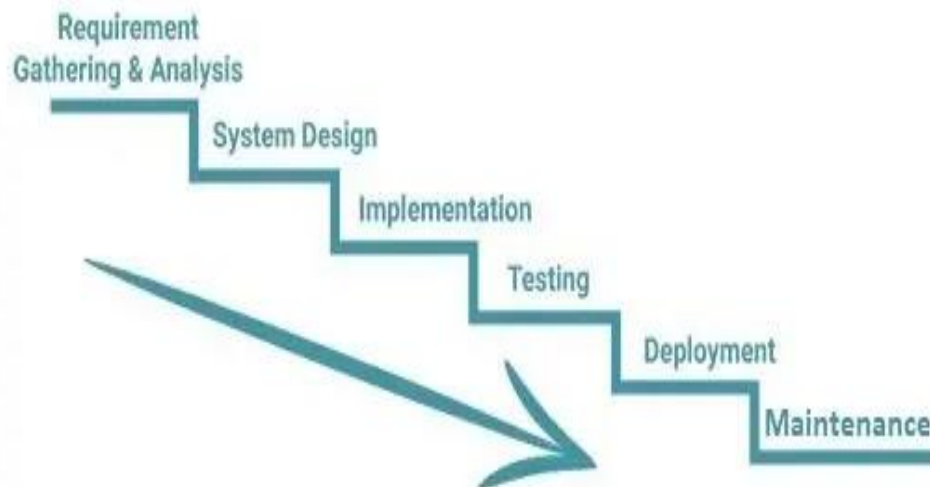


Figure 2:waterfall model

The Waterfall Model was first Process Model to be introduced. It is also referred to as a **linear-sequential life cycle model**. It is very simple to understand and use. In a waterfall model, each phase must be completed fully before the next phase can begin.

➤ Advantages and Disadvantages of Waterfall Model:

Advantages

- This model is simple and easy to understand and use.
- It is easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process.
- In this model phases are processed and completed one at a time. Phases do not overlap.
- Waterfall model works well for smaller projects where requirements are clearly defined and very well understood.

Disadvantages

- Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.
- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing

2. V model:

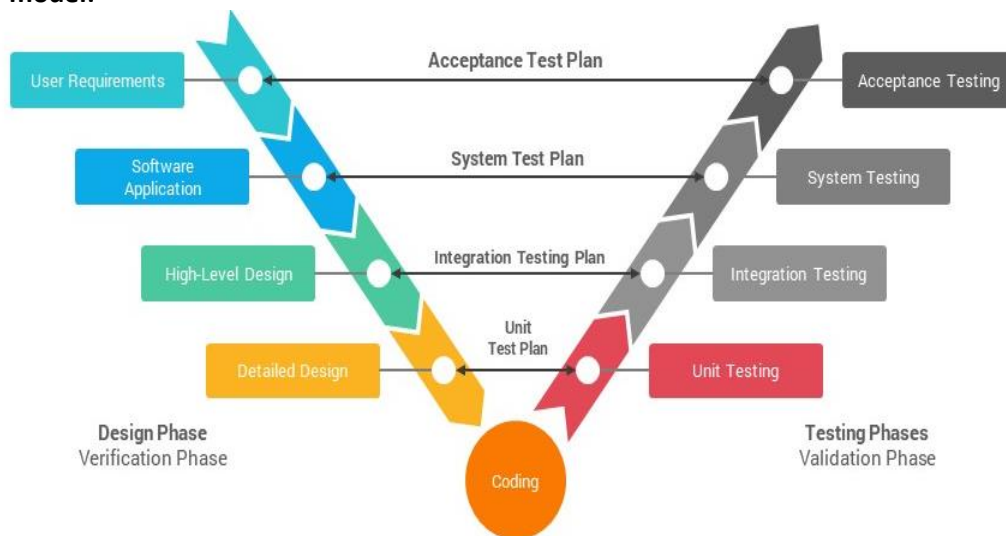


Figure 3: V model

V-model means Verification and Validation model. Just like the waterfall model, the V-Shaped life cycle is a sequential path of execution of processes. Each phase must be completed before the next phase begins. V-Model is one of the many software development models. Testing of the product is planned in parallel with a corresponding phase of development in V-model.

➤ Advantages and Disadvantages of V Model:

Advantages

- Simple and easy to use.
- Testing activities like planning, test designing happens well before coding. This saves a lot of time. Hence higher chance of success over the waterfall model.
- Proactive defect tracking – that is defects are found at early stage.
- Avoids the downward flow of the defects.
- Works well for small projects where requirements are easily understood.

Disadvantages

- Very rigid and least flexible.
- Software is developed during the implementation phase, so no early prototypes of the software are produced.
- If any changes happen in midway, then the test documents along with requirement documents must be updated.

3. Agile Model:

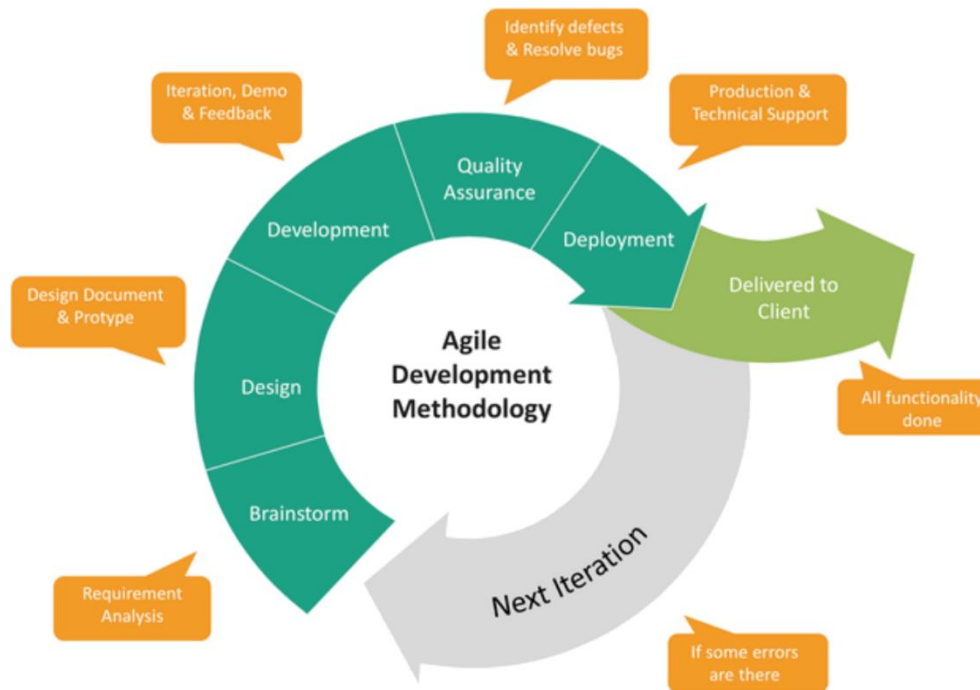


Figure 4:Agile Model

Agile development model is also a type of Incremental model. Software is developed in incremental, rapid cycles. This results in small incremental releases with each release building on previous functionality. Each release is thoroughly tested to ensure software quality is maintained. It is used for time critical applications.

Following are the Agile Manifesto principles –

- **Individuals and interactions** – In Agile development, self-organization and motivation are important, as are interactions like co-location and pair programming.
- **Working software** – Demo working software is considered the best means of communication with the customers to understand their requirements, instead of just depending on documentation.
- **Customer collaboration** – As the requirements cannot be gathered completely in the beginning of the project due to various factors, continuous customer interaction is very important to get proper product requirements.
- **Responding to change** – Agile Development is focused on quick responses to change and continuous development.

➤ **Advantages and Disadvantages of Spiral Model:**

Advantages

- Customer satisfaction by rapid, continuous delivery of useful software.
- Customers, developers and testers constantly interact with each other.
- Working software is delivered frequently (weeks rather than months).
- Close, daily cooperation between businesspeople and developers.
- Continuous attention to technical excellence and good design.
- Regular adaptation to changing circumstances.
- Even late changes in requirements are welcomed

Disadvantages

- In case of some software deliverables, especially the large ones, it is difficult to assess the effort required at the beginning of the software development life cycle.
- There is lack of emphasis on necessary designing and documentation.
- The project can easily get taken off track if the customer representative is not clear what outcome that they want.
- Only senior programmers are capable of taking the kind of decisions required during the development process. Hence it has no place for newbie programmers, unless combined with experienced resources.

Levels of testing:

Test levels in testing process depend on the size or functionality. Depending upon requirements there are different levels of testing.

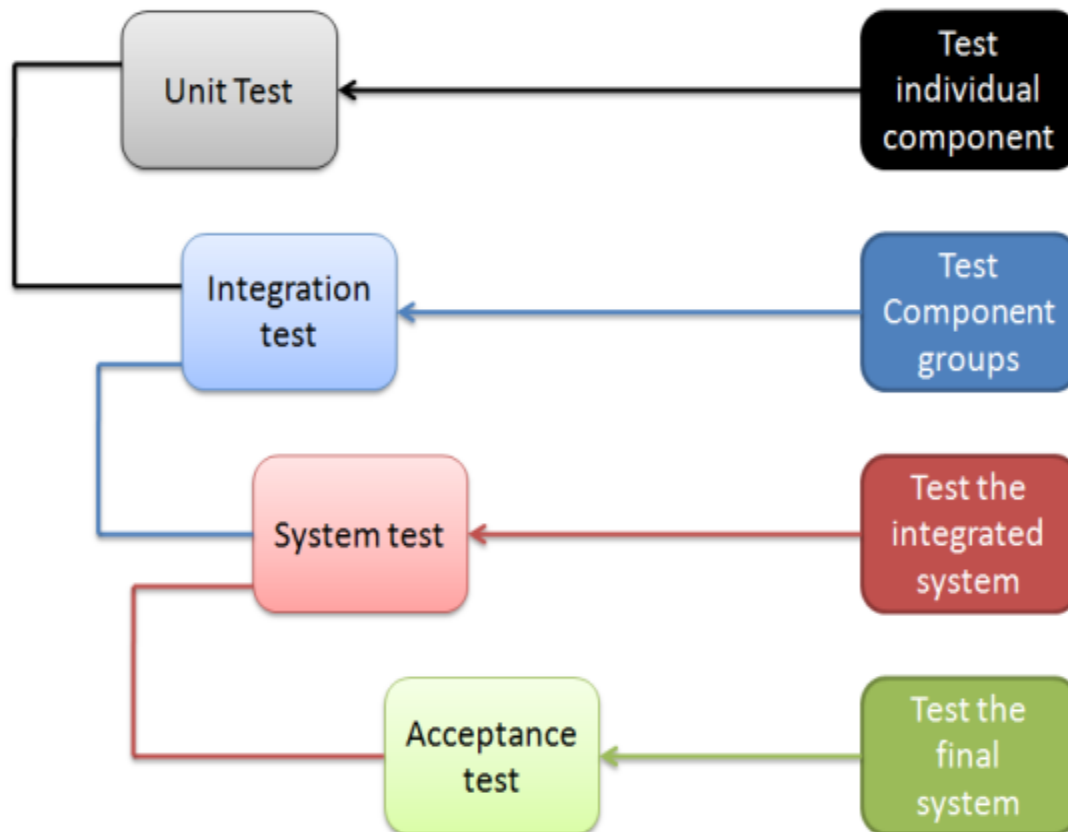


Figure 5:levels of testing

1. Component testing:

- As name suggests component testing is performed on separate component without integrating them into system.
- Component testing focuses on component and its functionality.
- Main objective of component testing is to find defects on component level. This helps to reduce the scope of faults and limit them on component level only.
- As defects in components are captured on component level only it helps to limit fault multiplication in system after integration.

2. Integration testing:

- Integration testing is done after combining all components into the system.
- It reduces the risk of failure due to integration.
- It verifies the functional and non-functional design.
- Integration testing also prevents faults to get into higher levels of integration.

3. System testing:

- System testing is done after integrating all the components into system.
- System testing is usually done to check overall functioning of system.
- It prevents the propagation of faults or defects into main functionality.

4. Acceptance testing:

- Acceptance testing is mainly done at users end or consumers end.
- An acceptance test can be understood to check if a previously defined “contract” between the developer and the customer is still on track. Running those acceptance tests also ensures that no requirement change has happened in the meantime and that everything is as it should be to satisfy the customer.

Types of testing

1. Functional testing:

- i. Functional testing focuses on verifying the main functionalities of object or software this totally depends upon the requirements.
- ii. Functional testing considers the behaviour of the software, hence black box testing is type of functional testing.

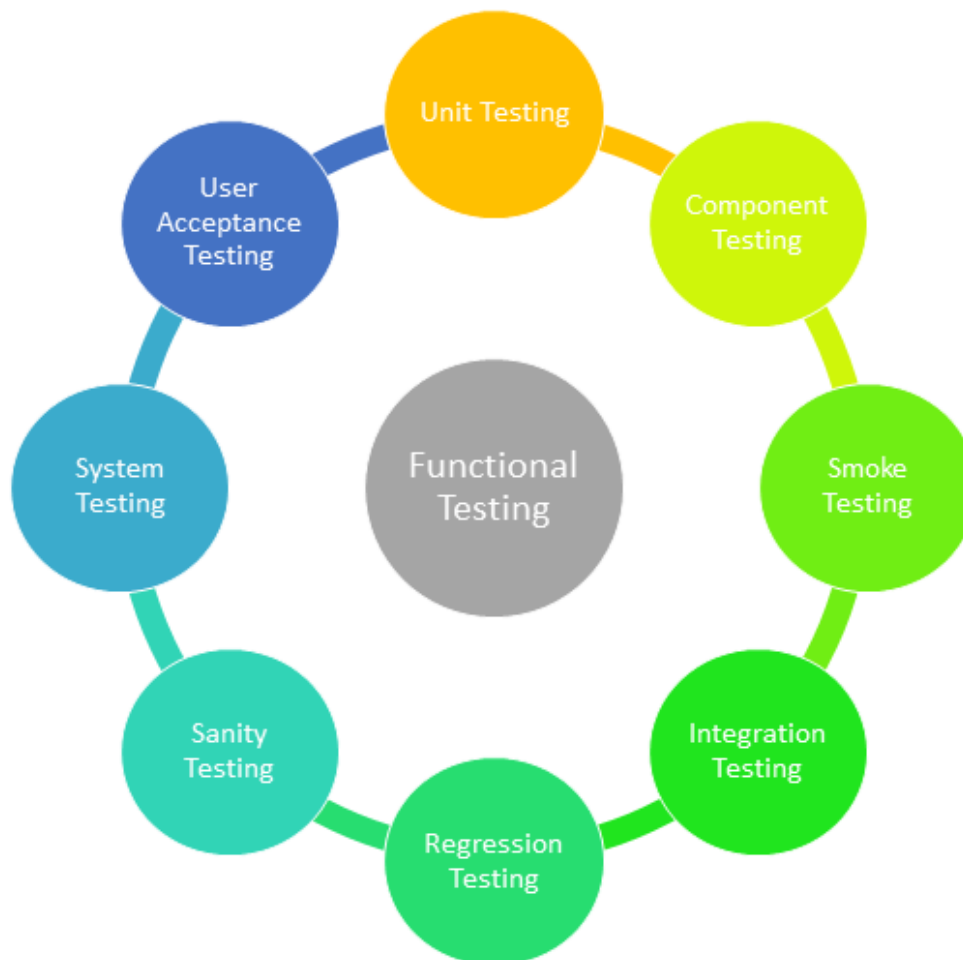


Figure 6:functional testing

2. Non-functional testing:

- i. Non-functional testing of a system evaluates characteristics of systems and software such as usability, performance efficiency or security.
- ii. Non-functional testing is the testing of “how well” the system behaves.
- iii. Black-box techniques may be used to derive test conditions and test cases for non-functional testing. For example, boundary value analysis can be used to define the stress conditions for performance tests.



Figure 6:Non-functional testing

Manual Testing:

There are two type of testing,

1. Static testing.
2. Dynamic testing.

➤ Difference between static and dynamic testing:

| Static testing | Dynamic testing |
|--|---|
| Done in verification stages | Done in validation testing stages |
| In static testing code is examined without execution. | In dynamic testing, code is executed. |
| Static testing is prevention of defects | Dynamic testing is about finding and fixing the defects. |
| Static testing gives assessment of code and documentation. | Dynamic testing gives bugs and defects in software in system. |
| Static testing can be performed before compilation. | Dynamic testing performed after compilation. |
| Static testing covers structural and functional coverage. | Dynamic testing covers executable file of program. |
| Static testing methods include walkthrough code review. | Dynamic testing methods involves functional and non-functional testing. |

Types of testcases:

Every testing has some testcases depending upon project. There are multiple types of test case. Testcases are classified as some specific type of cases.

Positive and negative test cases:

| positive testcases | Negative testcases |
|---|--|
| performed on system by provoking the valid data as a input. | Performed on the system by provoking invalid data as input |
| Checks whether an application behave as expected with the positive input. | Checks whether an application behaves as expected with negative input. |

Techniques used for designing the test cases:

| Techniques used | Explanation |
|-------------------------------|--|
| Equivalent partitioning | It divides data into partitions and verifies the working with that data. Nay partition can be divided into sub partitions. |
| Boundary value analysis | Boundary analysis checks the working of program with data representing the boundary or limit of processable data. |
| Decision table testing | Table of all possible paths is created, and each condition is checked. |
| Use case testing | Testcase are derived from use cases available |
| Decision testing and coverage | All the possible paths and code coverage is considered while testing. |

2.Activity:

Activity 1 –Requirements Issue

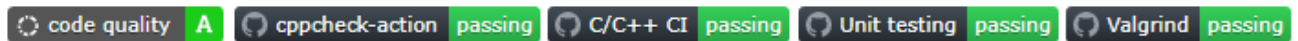
Task: Revisiting the requirements mentioned in the SDLC Calculator Project and comparing it with the implementation and raising issues if the requirements are not met.

Outcome: The requirements were checked and the corresponding issues were raised

Activity 2 – Checking Code Quality

Task: Explicit information on Code quality

Outcome: The code quality was checked using Codacy and the badge was obtained.



Activity 3 – Checking the Implementation of the Code

Task: Checking the implementation of code by breaking the code and running the make file.

Outcome: Checking whether the functions mentioned in the requirements were implemented and then the corresponding issues were raised.

Activity 4 – Solving of Issues

Task: The issues raised within the team and the peer review to be closed in the GitHub repository.





Outcome: The issues were labelled as High, Medium and Low based on the priorities. Most of the issues were resolved and the issues were closed. The readme file was updated.








Total No. of Issues raised: 63 issues

























No. of issues closed: 59 issues

No. of issues yet to be solved or to be done in future: 4 issues

| <input type="checkbox"/> 4 Open <input checked="" type="checkbox"/> 62 Closed | | Author ▾ | Label ▾ | Projects ▾ | Milestones ▾ | Assignee ▾ | Sort ▾ |
|---|---|----------|---------|------------|--------------|------------|--------|
| <input type="checkbox"/> | Low level requirements : percentage Low level Requirement Based #73 by 99003525 was closed 13 days ago | | | | | | |
| <input type="checkbox"/> | execution error High Level Requirement Based #72 by 99003525 was closed 13 days ago | | | | | | |
| <input type="checkbox"/> | Test plan : permutation and combination Medium Level Requirement Based #71 by 99003525 was closed 13 days ago | | | | | | |
| <input type="checkbox"/> | Test Cases High Level Requirement Based #70 by 99003525 was closed 13 days ago | | | | | | |
| <input type="checkbox"/> | UML diagrams Medium Level Requirement Based #69 by 99003525 was closed 13 days ago | | | | | | |
| <input type="checkbox"/> | low level requirements Medium Level Requirement Based #68 by 99003525 was closed 13 days ago | | | | | | |
| <input type="checkbox"/> | Overflow not handled Boundary Condition Based Medium Level #66 by 99003716 was closed 13 days ago | | | | | | |
| <input type="checkbox"/> | Factorial - Low level requirements Low level Requirement Based #64 by 99003525 was closed 13 days ago | | | | | | |
| <input type="checkbox"/> | Multiply - redundant lines of code Medium Level Requirement Based #63 by 99003525 was closed 13 days ago | | | | | | |
| <input type="checkbox"/> | Test Plan documentation of nPr and nCr Medium Level Requirement Based #62 by 99003525 was closed 13 days ago | | | | | | |
| <input type="checkbox"/> | inch to centimeter : error handling Low level Scenario Based #61 by 99003525 was closed 12 days ago | | | | | | |
| <input type="checkbox"/> | Factorial - input greater than 11 Boundary Condition Based Medium Level #60 by 99003525 was closed 13 days ago | | | | | | |
| <input type="checkbox"/> | Division - Floating point numbers Medium Level Requirement Based #59 by 99003525 was closed 13 days ago | | | | | | |
| <input type="checkbox"/> | Test plan not matching with test cases. Medium Level Scenario Based #57 by 99003518 was closed 20 days ago | | | | | | |
| <input type="checkbox"/> | Unit Testing and C/C++ are not passing Scenario Based #56 by 99003591 was closed 21 days ago | | | | | | |
| <input type="checkbox"/> | Dynamic code analysis failed. Analysis Based High Level #54 by 99003518 was closed 21 days ago | | | | | | |
| <input type="checkbox"/> | Naming of file Analysis Based Medium Level #53 by enskomali was closed 21 days ago | | | | | | |
| <input type="checkbox"/> | Test case Medium Level Scenario Based #52 by 99003519 was closed 21 days ago | | | | | | |
| <input type="checkbox"/> | Declaration of int Analysis Based Medium Level #51 by enskomali was closed 21 days ago TDLC | | | | | | |
| <input type="checkbox"/> | Division Operator Analysis Based Medium Level #50 by 99003524 was closed 21 days ago | | | | | | |
| <input type="checkbox"/> | Addition function test cases. Medium Level Scenario Based #49 by 99003518 was closed 20 days ago | | | | | | |
| <input type="checkbox"/> | Division Function test cases: Low level Scenario Based #48 by 99003518 was closed 21 days ago | | | | | | |
| <input type="checkbox"/> | Division Function Implementation Low level #47 by 99003518 was closed 21 days ago | | | | | | |
| <input type="checkbox"/> | subtraction function Analysis Based Medium Level #46 by 99003522 was closed 21 days ago | | | | | | |
| <input type="checkbox"/> | nCr and nPr function - negative input condition Analysis Based Medium Level #45 by 99003524 was closed 21 days ago | | | | | | |

| | | | | | |
|--------------------------|--|----------------|-------------------|---|---|
| <input type="checkbox"/> | Permutation and combination function | Analysis Based | Medium Level | | 2 |
| | #43 by 99003520 was closed 21 days ago | | | | |
| <input type="checkbox"/> | Percentage function | Analysis Based | Medium Level |  | 2 |
| | #42 by 99003522 was closed 21 days ago | | | | |
| <input type="checkbox"/> | TestPlan | Medium Level | Scenario Based | | 1 |
| | #41 by 99003519 was closed 21 days ago | | | | |
| <input type="checkbox"/> | Hexadecimal inputs | Analysis Based | Low level |  | 1 |
| | #40 by 99003522 was closed 21 days ago | | | | |
| <input type="checkbox"/> | Code Quality | High Level | | | 1 |
| | #39 by 99003519 was closed 21 days ago | | | | |
| <input type="checkbox"/> | Divided by zero condition in modulo operator | Analysis Based | Medium Level | | 1 |
| | #38 by 99003524 was closed 21 days ago | | | | |
| <input type="checkbox"/> | Bit operation function | Analysis Based | Low level |  | 1 |
| | #37 by 99003522 was closed 21 days ago | | | | |
| <input type="checkbox"/> | Readability | Medium Level | | | 1 |
| | #36 by enskomali was closed 21 days ago | | | | |
| <input type="checkbox"/> | Precision of calculator output is not specified | Medium Level | | | 1 |
| | #34 by 99003524 was closed 21 days ago | | | | |
| <input type="checkbox"/> | Implementation | Analysis Based | Medium Level | | 1 |
| | #33 by 99003520 was closed 21 days ago | | | | |
| <input type="checkbox"/> | Sequence of performing operations and inputs is not correct. | Analysis Based | Medium Level |  | 1 |
| | #32 by 99003518 was closed 20 days ago | | | | |
| <input type="checkbox"/> | Limited number of inputs | Medium Level | | | 1 |
| | #31 by 99003524 was closed 21 days ago | | | | |
| <input type="checkbox"/> | Requirements | High Level | Requirement Based | | |
| | #29 by 99003519 was closed 21 days ago | | | | |
| <input type="checkbox"/> | All possible functions are not implemented | Analysis Based | High Level | | 1 |
| | #28 by 99003524 was closed 21 days ago | | | | |

| | | | | | |
|--------------------------|--|--------------|-------------------|---|---|
| <input type="checkbox"/> | High level test plan | Low level | Scenario Based |  | 1 |
| | #27 by 99003522 was closed 21 days ago | | | | |
| <input type="checkbox"/> | Scope of testing. | Low level | Scenario Based |  | 1 |
| | #26 by 99003518 was closed 20 days ago | | | | |
| <input type="checkbox"/> | Incomplete Test plan. | Low level | Scenario Based |  | 1 |
| | #25 by 99003518 was closed 20 days ago | | | | |
| <input type="checkbox"/> | Low level Test plan | Low level | Scenario Based |  | 1 |
| | #24 by 99003522 was closed 21 days ago | | | | |
| <input type="checkbox"/> | Low level requirements | Low level | Requirement Based | | 1 |
| | #23 by 99003520 was closed 21 days ago | | | | |
| <input type="checkbox"/> | Test plan | Medium Level | Scenario Based | | 1 |
| | #22 by enskomali was closed 21 days ago | | | | |
| <input type="checkbox"/> | Test Plan not mentioned | High Level | Scenario Based | | 1 |
| | #20 by 99003524 was closed 21 days ago | | | | |
| <input type="checkbox"/> | Procedure to follow if multiple functions are used at the same time. | Medium Level | |  | 1 |
| | #19 by 99003518 was closed 20 days ago | | | | |
| <input type="checkbox"/> | High level requirements(functions) | High Level | Requirement Based |  | 1 |
| | #18 by 99003522 was closed 21 days ago | | | | |
| <input type="checkbox"/> | No specification for "valid input". | Medium Level | |  | |
| | #17 by 99003518 was closed 20 days ago | | | | |
| <input type="checkbox"/> | Test Plan | Low level | Scenario Based | | 1 |
| | #16 by 99003519 was closed 21 days ago | | | | |

| | | |
|--------------------------|---|---|
| <input type="checkbox"/> |  Improper Orientation Low level Requirement Based |  1 |
| | #15 by 99003524 was closed 21 days ago | |
| <input type="checkbox"/> |  Low Level Requirements: Inputs for each operations in not mentioned. Medium Level Requirement Based |  1 |
| | #14 by 99003518 was closed 20 days ago | |
| <input type="checkbox"/> |  Operator inputs and outputs can be in a different data type Medium Level Requirement Based |  1 |
| | #13 by 99003524 was closed 21 days ago | |
| <input type="checkbox"/> |  High level requirements Medium Level Requirement Based |  1 |
| | #12 by 99003520 was closed 21 days ago | |
| <input type="checkbox"/> |  Division function Analysis Based Medium Level |  1 |
| | #11 by 99003522 was closed 21 days ago | |
| <input type="checkbox"/> |  Low Level Requirements: Medium Level Requirement Based |  1 |
| | #10 by 99003518 was closed 20 days ago | |
| <input type="checkbox"/> |  Limitation in Main file Input Option High Level |  1 |
| | #9 by 99003522 was closed 21 days ago | |
| <input type="checkbox"/> |  High level requirements Medium Level Requirement Based |  1 |
| | #7 by 99003520 was closed 21 days ago | |
| <input type="checkbox"/> |  High level Requirements: Low level Requirement Based |  1 |
| | #5 by 99003518 was closed 20 days ago | |
| <input type="checkbox"/> |  Test Cases Scenario Based |  1 |
| | #4 by 99003524 was closed on Feb 6 | |
| <input type="checkbox"/> |  Code errors. |  1 |
| | #2 by 99003518 was closed on Feb 6 | |
| <input type="checkbox"/> |  uml diagrams issue |  2 |
| | #1 by 99003518 was closed on Feb 5 | |

Issues Closed (59 issues)

Filters

is:issue is:open

Labels8

Milestones1

New issue

4 Open

62 Closed

AuthorLabelProjectsMilestonesAssigneeSort

Scenario based

High Level

Scenario Based

#67 opened 13 days ago by 99003716

Modulo : floating point inputs

High Level

Scenario Based

#65 opened 13 days ago by 99003525

execution of Multiple operations at time

Medium Level

Scenario Based

#58 opened 13 days ago by 99003525

Addition function

Analysis Based

Medium Level

#30 opened 21 days ago by 99003522

Issues Yet to be Closed (4 issues)

GitHub link: https://github.com/99003518/Team2_calciapp