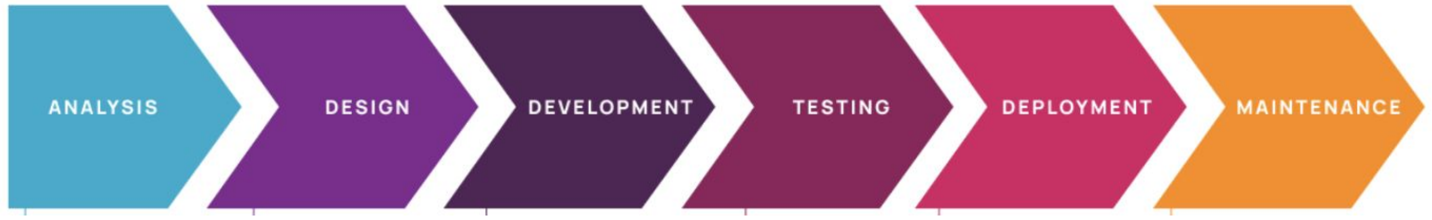
# STLC

Software Testing Life Cycle

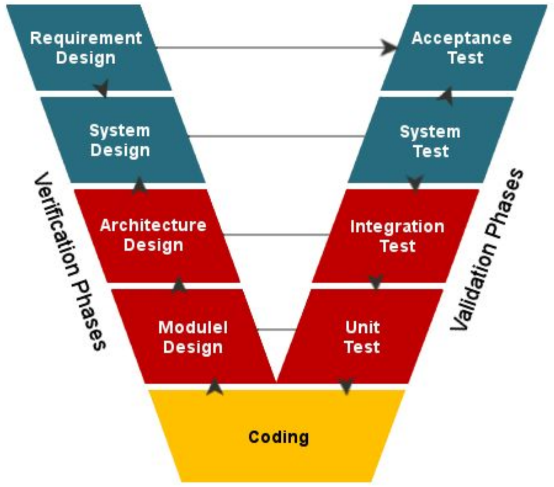
## SDLC

SDLC stands for Software Development Life Cycle, it provides a framework for efficient and effective software development, ensuring that the final product meets the requirements and objectives of the stakeholders.



## V Model

V Model, also called Verification and Validation model. Is a model that can help us control the process of SDLC.



PM -> Dev team lead -> URS -> Test team lead -> UAT Plan & Test Cases

Dev team lead -> SRS -> Test team lead -> System Testing Plan & Test Cases

Senior Dev -> MLD & Integration Testing Plan & Test Cases

Mid & Entry Level Dev -> LLD & Unit Testing Plan & Test Cases

## Verification vs Validation

### Verification

- Answers the question “Are we building the application in the right way according to specifications and requirement”
- Applis static testing which focus on the documentation (does not change often)
- Techniques in static testing
    - **Review**: requirement/design/test plan
    - **Walkthrough**: informal meeting with peers
    - **Inspection:** formal and scheduled meeting with peers

### Validation

- Answers the question “Are we building the right application?”
- Applies dynamic testing which focus on the application itself. (always changes)
- Techniques in dynamic testing
    - Unit Testing
    - Integration Testing
    - System Testing
    - UAT (User Acceptance Testing)

## Pros and Cons

**Pros:**

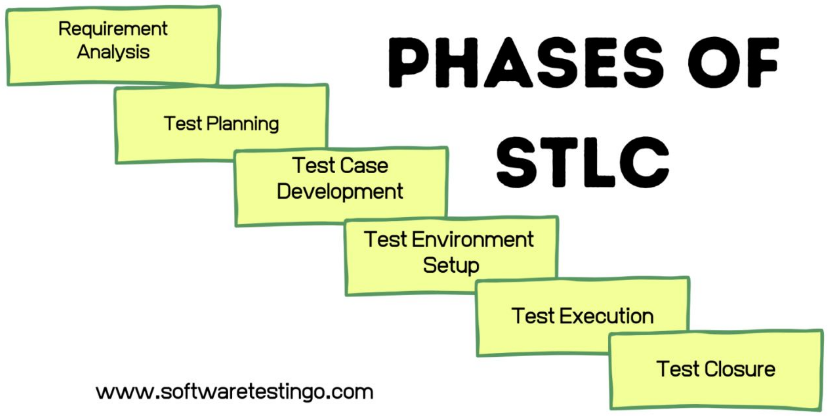
- Testing is involved in each and every phase of the project

**Cons:**

- Documentation is more
    - CRS (Customer Requirement Specification)
    - SRS (System Requirement Specification)
    - HLD (High Level Design aka System Design)
    - MLD (Mid Level Design)
    - LLD (Low Level Design)
- Initial investment is more (we need to hire both developers and tests at the initial phase)

## STLC

STLC stands for Software Testing Life Cycle, it is a part of SDLC and defines the steps and phases involved in the testing process and provides a framework for systematic testing activities.



## Requirement Analysis

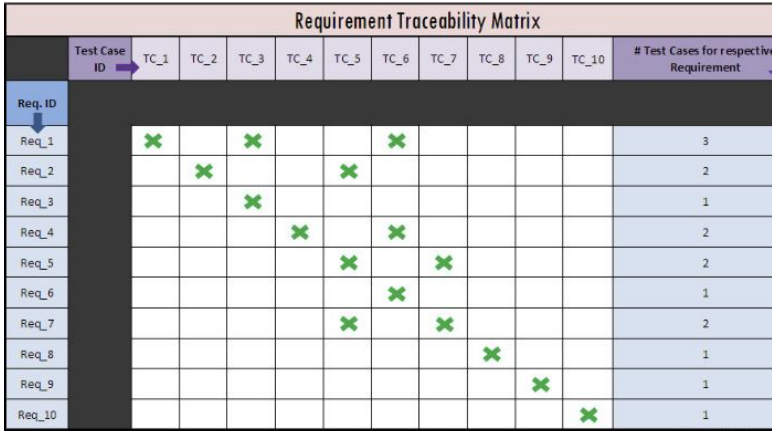
At the beginning of the test, the testing team should study the requirements (functional & non-functional ) from a testing point, they also need to communicate with various stakeholders to understand the requirements in detail.

### Activities

- Identify types of tests to be performed.
- Gather details about testing priorities and focus.
- Automation feasibility analysis
- Prepare Requirement Traceability Matrix (RTM)

### Output:

- RTM
- Automation feasibility analysis



Requirement Traceability Matrix

| Req. ID / Test Case ID | TC_1 | TC_2 | TC_3 | TC_4 | TC_5 | TC_6 | TC_7 | TC_8 | TC_9 | TC_10 | # Test Cases for respectiv Requirement |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Req_1 | ✖ | | ✖ | | | ✖ | | | | | 3 |
| Req_2 | | ✖ | | | ✖ | | | | | | 2 |
| Req_3 | | | ✖ | | | | | | | | 1 |
| Req_4 | | | | ✖ | | ✖ | | | | | 2 |
| Req_5 | | | | | ✖ | | ✖ | | | | 2 |
| Req_6 | | | | | | ✖ | | | | | 1 |
| Req_7 | | | | | ✖ | | ✖ | | | | 2 |
| Req_8 | | | | | | | | ✖ | | | 1 |
| Req_9 | | | | | | | | | ✖ | | 1 |
| Req_10 | | | | | | | | | | ✖ | 1 |

## Test Planning

**Input:**

- Project Plan (when to test)
- Functional/Non-functional  Requirements (what & how to test)

**Activities: (usually performed by team lead and senior testing engineers)**

- Resource Identification (Human & hardware resources)
- Test Estimation
- Preparation of Test Plan

**Output:**

- Test Plan

## Test case Development

**Input**

- Test Plan

**Activities**

- Create, verify and implement test cases & test scripts
- Prepare Test Data

**Output**

- Test case
- Test data

## Test Environment Set up

**Input:**

- Test case
- Test Data

**Activities:**

- Hardware/Software set up
- Environment setup and test data preparation
- Perform smoke test after build

**Output:**

- A ready environment with test data set up
- Smoke test results

## Test Execution

**Input:**

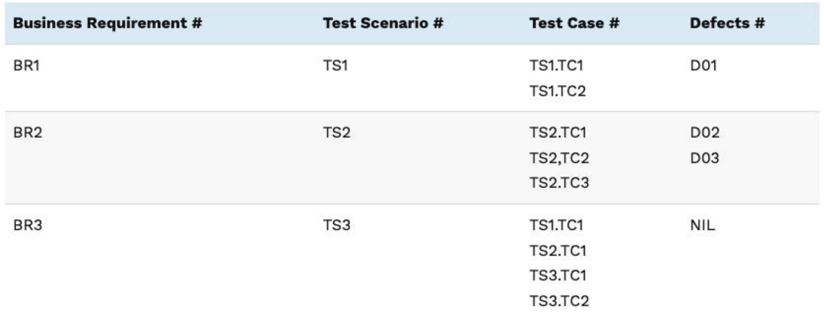
- Test environment set up
- Test cases & scripts

**Activities:**

- Test execution
- Document test results and log defect for failed test cases
- Map defects to test cases in RTM
- Retest the Defect fixes
- Track defect

Output:

- Completed RTM with execution status
- Defect Reports



| Business Requirement # | Test Scenario # | Test Case # | Defects # |
|---|---|---|---|
| BR1 | TS1 | TS1.TC1<br>TS1.TC2 | D01 |
| BR2 | TS2 | TS2.TC1<br>TS2,TC2<br>TS2.TC3 | D02<br>D03 |
| BR3 | TS3 | TS1.TC1<br>TS2.TC1<br>TS3.TC1<br>TS3.TC2 | NIL |

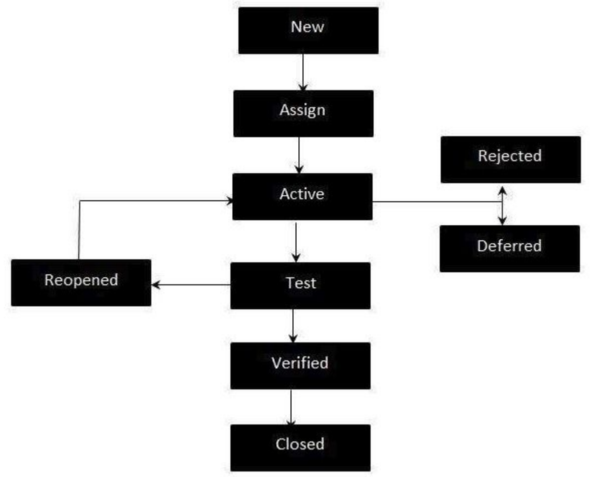
## Severity vs Priority

**Defect Severity** means how badly the defect has affected the application’s functionality.

**Defect Priority** defines the order in which developers will fix defects (because priority describes business importance). Note that the higher the impact of a bug on a business’s bottom line, the higher the priority assigned to it.

## Defect Life Cycle

- **New** - A potential defect is raised
- **Assigned** - Assigned to a developer
- **Active** - Defect is being addressed
- **Deferred** - When a defect cannot be addressed in the current cycle, it is deferred to future release
- **Rejected** - Duplicate defect, Not a defect, Not reproducible
- **Test** - The defect is fixed and ready for testing
- **Verified** - The defect is re tested and verified by QA
- **Closed** - final state
- **Reopened** - When the defect is not fixed



## Test Closure

Test Cycle Closure phase is completion of test execution which involves several activities like test completion reporting, collection of test completion matrices and test results. Testing team members meet, discuss and analyze testing artifacts to identify strategies that have to be implemented in future, taking lessons from current test cycle. The idea is to remove process bottlenecks for future test cycles.

**Activities**

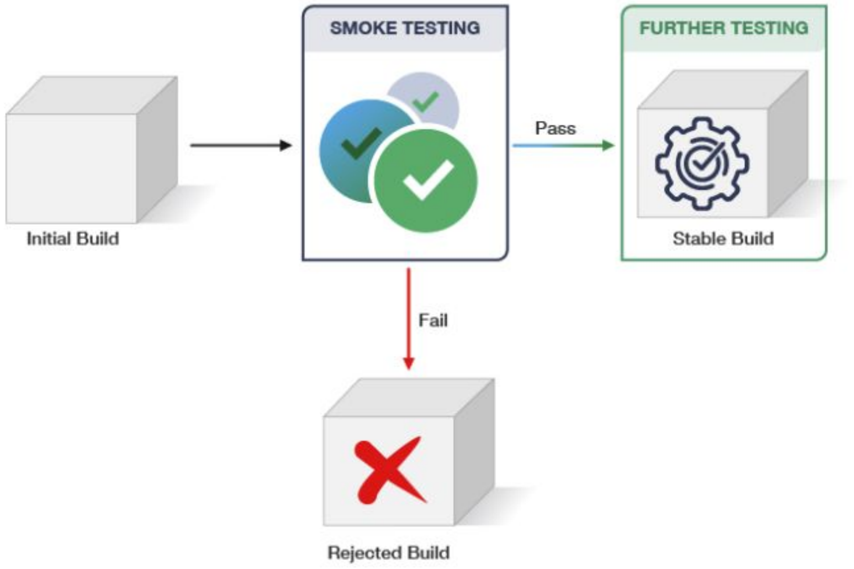
- Prepare test closure report
- Qualitative and quantitative reporting of quality of the work product to the customer.
- Test result analysis to find out the defect distribution by type and severity.

**Output:**

- Test closure report
- Test metrics

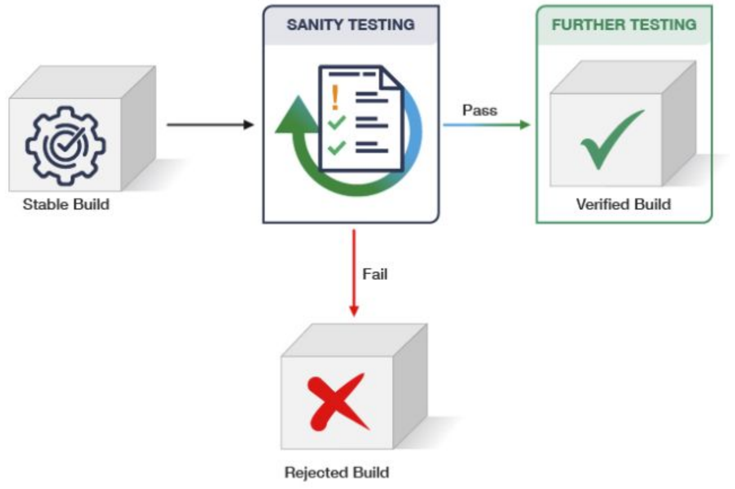
## Smoke Test

**Smoke testing:** is an approach which is usually carried out during the environment set up stages of the Software Development Life Cycle(SDLC) to make sure that the core functionalities of a program are working fine without any issues. It is executed before any detailed functional tests are done on the software. It is called called a **Build Verification Test**



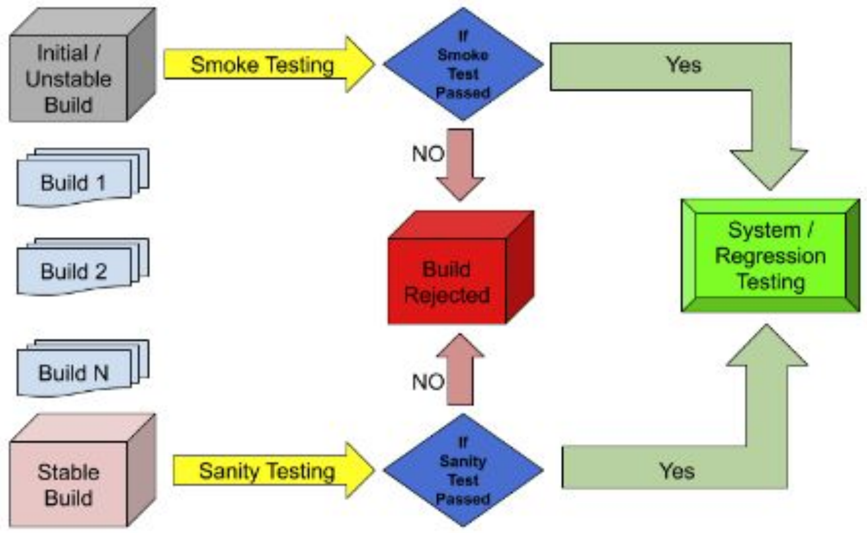
## Sanity Test

Sanity test is the subset of regression testing. it ensures that bug fixes, newly added functionality or any other code change doesn't affect the stable software, build other functionality or introduce any type of bug issue. It also evaluates the software and ensures whether it's ready for the next level of testing or not.

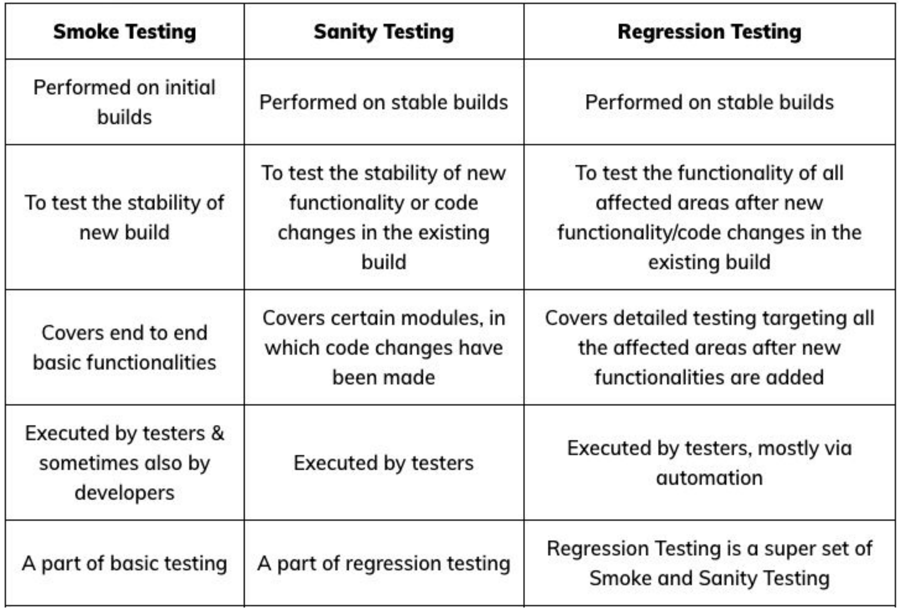


## Regression Test

**Regression testing** is the verification of “bug fixes or any changes in the requirement” and making sure they are not affecting other functionalities of the application. Regression testing is effective on automation and usually performed after some modifications have been made in the software build after requirement changes or bug fixes.



## Smoke vs Sanity vs Regression Test

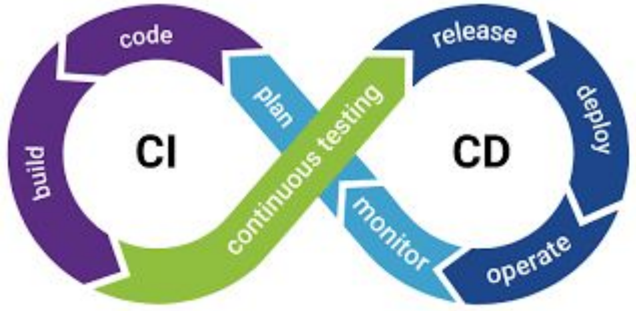


| Smoke Testing | Sanity Testing | Regression Testing |
|---|---|---|
| Performed on initial builds | Performed on stable builds | Performed on stable builds |
| To test the stability of new build | To test the stability of new functionality or code changes in the existing build | To test the functionality of all affected areas after new functionality/code changes in the existing build |
| Covers end to end basic functionalities | Covers certain modules, in which code changes have been made | Covers detailed testing targeting all the affected areas after new functionalities are added |
| Executed by testers & sometimes also by developers | Executed by testers | Executed by testers, mostly via automation |
| A part of basic testing | A part of regression testing | Regression Testing is a super set of Smoke and Sanity Testing |

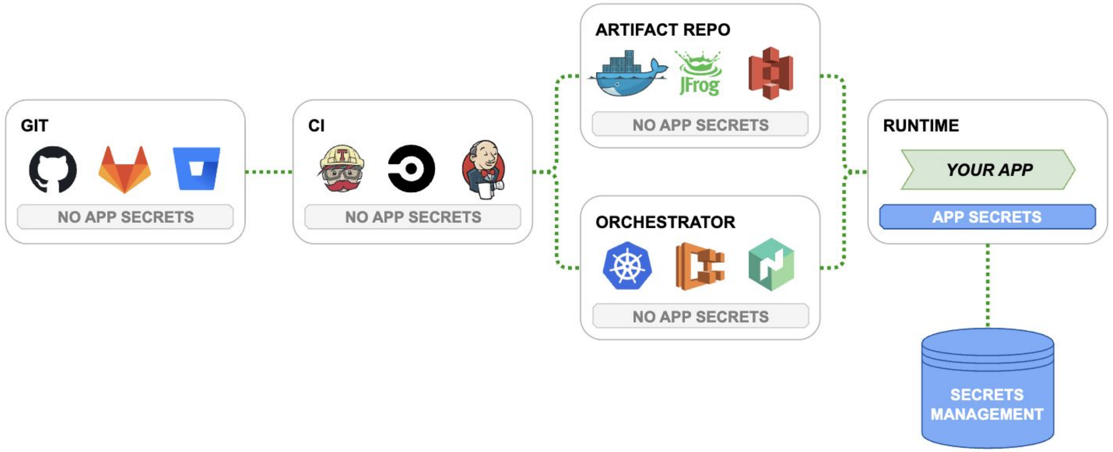
## CI/CD

**Continuous integration** is the practice of integrating all your code changes into the main branch of a shared source code repository early and often, automatically testing each change when you commit or merge them, and automatically kicking off a build.

**Continuous delivery**
Once code has been tested and built as part of the CI process, CD takes over during the final stages to ensure it's packaged with everything it needs to deploy to any environment at any time. CD can cover everything from provisioning the infrastructure to deploying the application to the testing or production environment.



## Real Life Example


GIT
NO APP SECRETS
CI
NO APP SECRETS
ARTIFACT REPO
JFrog
NO APP SECRETS
ORCHESTRATOR
NO APP SECRETS
RUNTIME
YOUR APP
APP SECRETS
SECRETS MANAGEMENT